Building a DEC GKS Workstation Handler System

Order Number: AA-MJ34A-TE

April 1989

This guide explains how to write GKS workstation handlers that DIGITAL does not directly support. Chapter 1, Introduction, and Chapter 2, How GKS Works, provide background information about DEC GKS, and the remainder of the book explains how to build a full workstation handler.

Operating System and Version:	VMS Version 4.7 or higher. ULTRIX Version 3.0 or higher. VAXstation requirement: VAXstation Windowing Software Versions 3.3 or higher, or DECwindows Version 1.0.
Software Version:	DEC GKS Version 4.0

digital equipment corporation
maynard, massachusetts

First printing, May 1986
Revised March 1987, April 1989

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.

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

The following are trademarks of Digital Equipment Corporation:

ALL-IN-1	EduSystem	RT
DEC	IAS	ULTRIX
DEC/CMS	MASSBUS	UNIBUS
DEC/MMS	PDP	VAX
DECnet	PDT	VAXcluster
DECmate	P/OS	VMS
DECsystem-10	Professional	VT
DECSYSTEM-20	Q-bus	Work Processor
DECUS	Rainbow	
DECwriter	RSTS	
DIBOL	RSX	digital™

ZK4634

Contents

Preface xiii

Summary of Technical Changes xv

Chapter 1 Introduction

1.1 Writing Handlers for Your Devices 1–2

1.2 Workstation Handlers 1–3

Chapter 2 How GKS Works

2.1 The GKS System 2–1
2.1.1 Inquiries 2–1
2.1.2 Transformations 2–2
2.1.3 Input 2–4
2.1.4 Output 2–4

2.2 Communication Between the Kernel and Your Handler 2–5
2.2.1 DEC GKS Data Types 2–6
2.2.2 Parameter Passing 2–6

Chapter 3 Building a Workstation Handler System

3.1 Required Capabilities 3–2

3.2 Hardcopy Output Devices 3–3

3.3 Workstation Handler Data Structures 3–4
3.3.1 Building the Workstation Description Table 3–4

3.4 Workstation State List 3–25

3.5 Required Functions 3–32
3.5.1 Segment Simulation 3–32
3.5.2 Message and Constant Files 3–33

3.6 Suggested Escape Function 3–34
3.6.1 INQUIRE GDP EXTENT 3–34

3.7 Developing Your Device Function Table 3–35

3.8 Linking Your Handler to DEC GKS 3–44

3.9 Defining Workstation Handler Logical Names 3–44
3.9.1 Handler Logical Names 3–45
3.9.2 Adding Logical Names to GKSTARTUP.COM 3–48
3.9.3 Reentrance 3–48

Chapter 4 Workstation Handler Control and Transformation Functions

4.1 Active Attribute Array 4–1

4.2 Function Descriptions 4–3
OPEN WORKSTATION 4–4
CLOSE WORKSTATION 4–6
CLEAR WORKSTATION 4–7
UPDATE WORKSTATION 4–9
PERFORM DEFERRED OUTPUT 4–11
ESCAPE 4–12
SET WORKSTATION WINDOW 4–14
SET WORKSTATION VIEWPORT 4–16
SET NORMALIZATION TRANSFORMATION 4–18
SET DEFERRAL MODE 4–20

REDRAW ALL SEGMENTS ON WORKSTATION 4–22
SET GLOBAL INTERACTIONS 4–23
MESSAGE 4–25
SET NDC TRANSFORMATION 4–26

Chapter 5 Workstation Handler Input Functions

5.1 Writing Input Functions 5–1
5.1.1 INITIALIZE Functions 5–2
5.1.2 SET Functions 5–2

5.2 REQUEST, SAMPLE, and EVENT Input 5–2
5.2.1 Managing SAMPLE and EVENT Input 5–3
5.2.2 GKS$STORE_EVENTS 5–4

5.3 Function Descriptions 5–6
INITIALIZE LOCATOR 5–7
INITIALIZE STROKE 5–13
INITIALIZE VALUATOR 5–18
INITIALIZE CHOICE 5–21
INITIALIZE STRING 5–25
INITIALIZE PICK 5–28
SET LOCATOR MODE 5–31
SET STROKE MODE 5–33
SET VALUATOR MODE 5–35
SET CHOICE MODE 5–37
SET STRING MODE 5–39
SET PICK MODE 5–41
REQUEST LOCATOR 5–43
REQUEST STROKE 5–45
REQUEST VALUATOR 5–47
REQUEST CHOICE 5–49
REQUEST STRING 5–51
REQUEST PICK 5–53
SAMPLE LOCATOR 5–55
SAMPLE STROKE 5–56
SAMPLE CHOICE 5–58
SAMPLE VALUATOR 5–60
SAMPLE STRING 5–61
SAMPLE PICK 5–63

Chapter 6 Workstation Handler Inquiry Functions

INQUIRE LIST OF POLYLINE INDEXES 6–3
INQUIRE POLYLINE REPRESENTATION 6–5
INQUIRE LIST OF POLYMARKER INDEXES 6–7
INQUIRE POLYMARKER REPRESENTATION 6–9
INQUIRE LIST OF TEXT INDEXES 6–11
INQUIRE TEXT REPRESENTATION 6–13
INQUIRE TEXT EXTENT 6–15
INQUIRE LIST OF FILL AREA INDEXES 6–25
INQUIRE FILL AREA REPRESENTATION 6–27
INQUIRE LIST OF PATTERN INDEXES 6–29
INQUIRE PATTERN REPRESENTATION 6–31
INQUIRE LIST OF COLOR INDEXES 6–33
INQUIRE COLOR REPRESENTATION 6–35
INQUIRE WORKSTATION TRANSFORMATION 6–37
INQUIRE LOCATOR DEVICE STATE 6–39
INQUIRE STROKE DEVICE STATE 6–42
INQUIRE VALUATOR DEVICE STATE 6–45
INQUIRE CHOICE DEVICE STATE 6–47
INQUIRE STRING DEVICE STATE 6–50
INQUIRE PICK DEVICE STATE 6–52
INQUIRE WORKSTATION DEFERRAL AND UPDATE STATE 6–55
INQUIRE PIXEL ARRAY DIMENSIONS 6–57
INQUIRE PIXEL ARRAY 6–60
INQUIRE PIXEL 6–62
INQUIRE SEGMENT NAMES ON WORKSTATION 6–64
INQUIRE WORKSTATION CATEGORY 6–66
INQUIRE WORKSTATION CLASSIFICATION 6–68
INQUIRE DISPLAY SPACE SIZE 6–69
INQUIRE POLYLINE FACILITIES 6–71
INQUIRE PREDEFINED POLYLINE REPRESENTATION 6–73
INQUIRE POLYMARKER FACILITIES 6–75
INQUIRE PREDEFINED POLYMARKER REPRESENTATION 6–77
INQUIRE TEXT FACILITIES 6–79
INQUIRE PREDEFINED TEXT REPRESENTATION 6–81
INQUIRE FILL AREA FACILITIES 6–83
INQUIRE PREDEFINED FILL AREA REPRESENTATION 6–85
INQUIRE PATTERN FACILITIES 6–87
INQUIRE PREDEFINED PATTERN REPRESENTATION 6–88
INQUIRE COLOR FACILITIES 6–90
INQUIRE PREDEFINED COLOR REPRESENTATION 6–91
INQUIRE GDP PRIMITIVES 6–92
INQUIRE GENERALIZED DRAWING PRIMITIVE 6–94

INQUIRE MAXIMUM LENGTH OF WORKSTATION STATE TABLES ... 6–96
INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES 6–98
INQUIRE DEFAULT LOCATOR DEVICE DATA 6–100
INQUIRE DEFAULT STROKE DEVICE DATA 6–102
INQUIRE DEFAULT VALUATOR DEVICE DATA 6–104
INQUIRE DEFAULT CHOICE DEVICE DATA 6–106
INQUIRE DEFAULT STRING DEVICE DATA 6–108
INQUIRE DEFAULT PICK DEVICE DATA 6–110
INQUIRE DYNAMIC MODIFICATION OF WORKSTATION ATTRIBUTES 6–112
INQUIRE DEFAULT DEFERRAL STATE VALUES 6–114
INQUIRE NUMBER OF SEGMENT PRIORITIES SUPPORTED 6–116
INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES 6–117
INQUIRE SIZE OF HANDLER STORAGE 6–119

Chapter 7 Workstation Handler Metafile Functions

WRITE ITEM TO METAFILE 7–3
GET ITEM FROM METAFILE 7–5
READ ITEM FROM METAFILE 7–7

Chapter 8 Workstation Handler Set Representation Functions

8.1 Function Descriptions 8–2
SET POLYLINE REPRESENTATION 8–3
SET POLYMARKER REPRESENTATION 8–5
SET TEXT REPRESENTATION 8–7
SET FILL AREA REPRESENTATION 8–9
SET PATTERN REPRESENTATION 8–11
SET COLOR REPRESENTATION 8–13

Chapter 9 Workstation Handler Output Functions

9.1 Coordinate Data 9–1

9.2 Attributes 9–2

9.3 Aspect Source Flags 9–2

9.4 Segment Overlap 9–5

9.5 **Output Function Descriptions** 9–5
POLYLINE 9–6
POLYMARKER 9–8
TEXT 9–10
FILL AREA 9–15
CELL ARRAY 9–19
GDP 9–22
HIGHLIGHT EXTENT 9–24

Chapter 10 Workstation Handler Segment Functions
CREATE SEGMENT 10–2
CLOSE SEGMENT 10–3
RENAME SEGMENT 10–4
DELETE SEGMENT 10–5
SET SEGMENT TRANSFORMATION 10–7
SET VISIBILITY 10–9
SET SEGMENT PRIORITY 10–11
SET DETECTABILITY 10–13
SET HIGHLIGHTING 10–14

Appendix A Transformations

A.1 Concatenating Transformation Matrixes A–1

A.2 NDC Transformation and Segment Simulation A–2

A.3 NDC Transformation When Your Handler Supports Segments A–4

A.4 Algorithms for Transformations A–6
A.4.1 Transformations Assuming an Identity NDC Transformation A–6
A.4.2 Transformations Assuming the Nonidentity NDC Transformation A–8
A.4.2.1 Transforming from NDC to LDC A–8
A.4.2.2 Transforming from WC to LDC A–9

Appendix B Stroke Text Simulation Routines
GKS$SIM_STROKE_TEXT B–2
GKS$SIM_STROKE_TEXT_EXTENT B–6
GKS$SIM_STROKE_INQ_TEXT_FAC B–9

Appendix C Pick Simulation Functions
GKS$FIND_SEGMENT C–2
GKS$FIND_SEG_EXTENT C–6

Appendix D Workstation Handler Function Examples

D.1 Data Structures D–1

D.2 Control Functions D–8

D.3 Transformation Functions D–16

D.4 Output Functions D–21

D.5 Output Attribute Functions D–26

D.6 Inquiry Functions D–28

D.7 DFT Building Macro D–32

D.8 Linking Command Procedure D–34

Index

Examples
3–1 Sample DFT-Building Macro 3–42

Figures

1–1	The Application, GKS Kernel, and Handlers	1–2
1–2	Workstation Handler Diagram	1–4
6–1	Text Extents 1	6–17
6–2	Text Extents 2	6–18
6–3	Text Extents 3	6–19
6–4	Text Extents 4	6–20
6–5	Text Extents 5	6–21
6–6	Pixel Array Dimensions	6–58
A–1	Transformation Pipeline for Segment Simulation	A–3
A–2	Transformation Pipeline for Handlers that Support Segments	A–5

Tables

2–1	GKS State Lists and Description Tables	2–2
2–2	GKS Coordinate Systems	2–3
2–3	DEC GKS Data Types	2–6
2–4	Passing Mechanisms for DEC GKS Data Types	2–7
3–1	GKS Level 2c Required Capabilities	3–2
3–2	Workstation Handler Workstation Description Table Structure	3–5
3–3	WDT Items for All Workstation Types Except MI and MO	3–6
3–4	WDT Items for OUTPUT and OUTIN Workstations	3–6
3–5	WDT Items for LOCATOR Logical Input Devices	3–21
3–6	WDT Items for STROKE Logical Input Devices	3–22
3–7	WDT Items for VALUATOR Logical Input Devices	3–23
3–8	WDT Items for CHOICE Logical Input Devices	3–23
3–9	WDT Items for PICK Logical Input Devices	3–24
3–10	WDT Items for STRING Logical Input Devices	3–24
3–11	WSL Elements for All Workstation Types	3–26
3–12	WSL Elements for OUTIN, OUTPUT, and MO Workstations	3–26
3–13	WSL Items for LOCATOR Logical Input Devices	3–28
3–14	WSL Items for STROKE Logical Input Devices	3–29
3–15	WSL Items for VALUATOR Logical Input Devices	3–30
3–16	WSL Items CHOICE Logical Input Devices	3–30
3–17	WSL Items PICK Logical Input Devices	3–31
3–18	WSL Items for STRING Logical Input Devices	3–31

3–19 GKSMSGS and GKSDEFS File Extensions 3–33
3–20 Keywords for DFT Macro 3–36
3–21 Keywords for DFT_INPUT Macro 3–38
3–22 Keywords for DFT_GKS_INQ Macro 3–39
3–23 Keywords for DFT_WS_INQ Macro 3–40
4–1 Active Attribute Array Structure 4–2
4–2 Call Back Table Output Parameter 4–5
7–1 Required Metafile Format 7–2
7–2 Metafile Format Items Determined by Your Functions 7–2
9–1 ASF Bits 9–3
9–2 No Change Bits 9–3
9–3 GKS Bitmask Constants 9–4

Preface

This book explains how to build DEC GKS workstation graphics handlers.

Document Structure

Chapter 1, Introduction, and Chapter 2, How GKS Works, provide background information about DEC GKS. Review these chapters for an overview of DEC GKS before you write a handler. If you are unfamiliar with any of the concepts in these chapters, refer to the *DEC GKS Reference Manual.*

The remainder of this book explains how to build a workstation handler system. The workstation handler is the most efficient way to implement DEC GKS on your device, but it is far bigger and more complex than the device handler system. Chapter 3, Building a Workstation Handler System, describes data structures normally contained in a workstation handler system, as well as how to build and link the system. Chapter 4, Workstation Handler Control and Transformation Functions, through Chapter 10, Workstation Handler Segment Functions, describe the functions you must write to implement a workstation handler.

The appendixes provide additional information about transformations and built-in simulation routines for workstation handler systems, as well as example workstation handler functions.

Intended Audience

This book is intended for experienced systems programmers with a strong background in graphics. Familiarity with DEC GKS is assumed.

Associated Documents

This manual is part of the DEC GKS documentation set that includes the following books:

- The *DEC GKS Reference Manual*
- The *DEC GKS User Manual*
- The *DEC GKS C Binding Reference Manual*
- The *DEC GKS FORTRAN Binding Reference Manual*
- The *DEC GKS GKS$ Binding Reference Manual*
- The *DEC GKS Device Specifics Reference Manual*
- The *Building a DEC GKS Device Handler System*
- The *Building a DEC GKS Workstation Handler System*
- The *DEC GKS Installation Guide*

The reader should have a thorough understanding of the *DEC GKS Reference Manual.*

In addition, to develop a GKS implementation that conforms to the national standard, the reader should be familiar with the ANSI GKS standard, X3.124, available from the American National Standards Institute, Inc. The descriptions of GKS functions and data structures in this book are based on the ANSI standard.

Summary of Technical Changes

New and Changed Features

This is a new DEC GKS Version 4.0 manual, replacing in part the Version 3.0 manual, *Writing VAX GKS Graphics Handlers,* which is divided into two manuals for Version 4.0: *Building a DEC GKS Device Handler System* and *Building a DEC GKS Workstation Handler System.* Changes and additions to these manuals are minor.

The following is a change from Version 3.0 to Version 4.0 related to building a workstation graphics handler:

- The funtion OPEN WORKSTATION in Chapter 4, Workstation Handler Control and Transformation Functions, has a new parameter, CALL_BACK_TABLE.

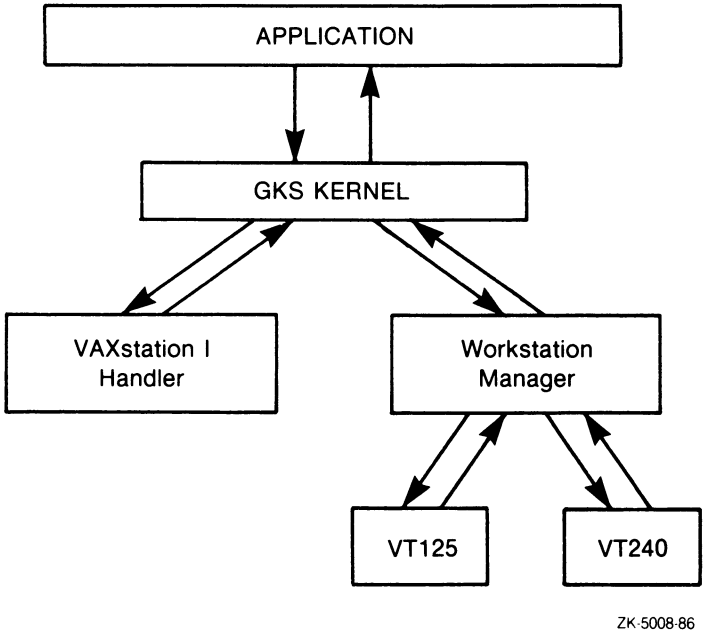
Chapter 1

Introduction

The DEC Graphical Kernel System (GKS) is a run-time library of graphics routines. A GKS implementation consists of a kernel and at least one handler. The kernel interprets calls to the GKS$ interface or a binding (that is, GKS functions as explained in the *DEC GKS Reference Manual*), and performs device-independent operations. Kernel operations include some inquiry functions, maintaining certain state tables, issuing calls to handlers, and some simulation routines.

Handlers perform device-specific operations. Handler operations include performing output, getting input, and responding to inquiries for workstation-specific information. You use a different handler for each type of workstation. The kernel can work with multiple handlers to drive many different devices simultaneously, as shown in Figure 1–1.

Figure 1–1: The Application, GKS Kernel, and Handlers



This chapter explains the process of writing handlers for devices that DIGITAL does not directly support.

1.1 Writing Handlers for Your Devices

You can write a handler to support any graphics device that DIGITAL does not support, then use those devices for GKS operations with your system. By writing a custom handler for your specific device, you can take full advantage of the features and capabilities available in your workstation.

Writing a handler involves writing graphics functions, and creating data structures and tables that serve as interfaces between your handler and the GKS kernel.

DEC GKS provides two methods of writing handlers. You can either write a device handler or a workstation handler. The following section describes how to write workstation handlers only. For information about writing device handlers, see the *Building a DEC GKS Device Handler System.*

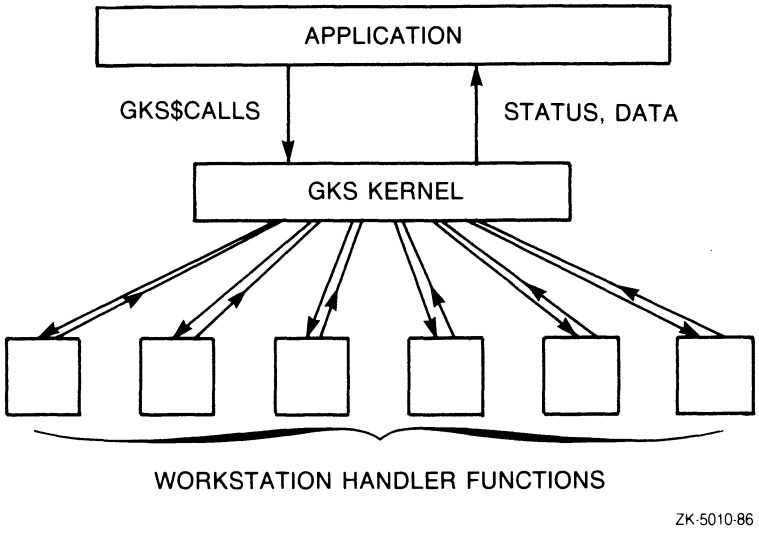
1.2 Workstation Handlers

The steps for writing a full workstation handler parallel writing a device handler except that you must provide approximately 125 routines and build several data structures. The routines include the following:

- Control operations
- Input functions
- Setting output attributes
- Output functions
- Transformations
- Inquiries
- Metafile functions (for metafile handlers)
- Segment operations (optional)

Figure 1–2, Workstation Handler Diagram, illustrates the operation of a workstation handler system.

Figure 1–2: Workstation Handler Diagram



When the GKS kernel receives a GKS command it interprets the command. If the kernel can perform the function by itself (for example, returning information from a table internal to the kernel), it performs the function and returns the result to the calling process.

If the kernel cannot perform the function specified by the GKS command, it executes a user-written function. At the completion of this process the function returns a result to the GKS kernel for eventual delivery to the calling process.

The advantage of a workstation handler is increased performance gained by taking advantage of the device's specific capabilities. The disadvantage is the longer time needed to implement the handler.

If your workstation has a large amount of intelligence or features, including complete hardware support for segments and pick input, and time-to-implementation is not a consideration, you should probably write a workstation handler. Another situation where you might choose to write a workstation handler is to support a metafile handler.

On the other hand, if you need to implement GKS as soon as possible on a device DIGITAL does not support directly, or if your device does not have significant intelligence or functionality, you should probably write a device handler. You can write a workstation handler after the device is running with a device handler system.

See Appendix D, Workstation Handler Function Examples, for more information about developing full workstation handlers.

Chapter 2

How GKS Works

This chapter provides the background information you need to develop GKS graphics handlers. It explains how GKS works, and the communications between the GKS kernel and the handlers.

2.1 The GKS System

The GKS system performs graphics output, accepts input on a graphics workstation, and performs inquiries and transformations. The following sections describe each of these operations. Note that these sections describe how the kernel interacts with a handler. If you are developing a workstation handler system you must provide functions to support each interaction.

2.1.1 Inquiries

GKS inquiries give you information from the state lists and description tables shown in Table 2-1.

Table 2–1: GKS State Lists and Description Tables

Name	Contents
GKS Description Table	Constant information about the GKS utility, including the implementation level and maximum number of transformations. This is a read-only structure contained in the kernel.
GKS State List	The current state of GKS, such as the list of open workstations, current normalization transformation number, and the value of each output attribute. This is a read/write structure maintained by the kernel.
Segment State List	The current state of each open segment, including the segment name, the workstations with which it is associated, and its priority and visibility. This is a read/write structure maintained by the kernel.
Workstation Description Table	Constant information about each workstation, including its category and its input and output capabilities. This is a read-only structure contained in the handler.
Workstation State List	Information about each open workstation's current input and output attributes. This is a read/write structure. Some of its information is maintained jointly by the kernel and the handler or workstation manager.

Additional GKS inquiries get information about picture elements (pixels) on an output device.

Most inquiry functions operate by looking up the requested information in the appropriate table. In device handler implementations, the workstation manager supplied by DIGITAL responds to most inquiry operations. Your device handler need only respond to the inquiries described in Chapter 4.

In workstation handler systems, you must provide functions to respond to 20 inquiries into the WSL, 23 inquiries into the WDT, and 3 pixel inquiries.

2.1.2 Transformations

Transformations let you map coordinates from one GKS coordinate system to another. In a device handler system, the workstation manager performs all transformations. In a workstation handler system, you must provide functions to perform transformations.

GKS uses three coordinate systems, as shown in Table 2-2.

Table 2–2: GKS Coordinate Systems

Coordinate System	Description
World Coordinates (WC)	A coordinate system defined by an X-axis and a Y-axis which intersect at 0 on each axis and continue to infinity in both the positive (up or right) and negative (down or left) direction. You define the portion of world coordinate space in which you will work by specifying the X and Y coordinates of the corners of a rectangle. The rectangle you specify is called the *world window*. Points in WC space are represented as real numbers.
Normalized Device Coordinates (NDC)	An imaginary display system with decimal coordinates from 0 to infinity on both the X and Y axis. Only the portion from 0 to 1 on each axis is visible. You map the world window to a portion of NDC space called the *world viewport*. Then you specify the portion of NDC space you want to use on a particular workstation. This is called the *workstation window*.
Logical Device Coordinates (LDC)	A coordinate system with the origin based in the lower left corner and assuming square units. The units may be based on the unit size of the device's native coordinate system. You map the workstation window to a portion of LDC space called the *workstation viewport*. Your device's specifications should list its coordinate range.

In addition to logical device coordinates, GKS also uses device coordinates (DC). The device coordinate system is the coordinate system your device understands. It may be integer or real numbers, and the origin may be at any point on the display surface. If your device uses a different origin, integer device coordinates, or nonsquare coordinates, you must convert your device coordinates to LDC. Your handler functions must also convert points from LDC to DC for output functions.

The multiple coordinate systems let GKS operate independently of the size or X/Y aspect ratio of individual workstations. You can generate graphics images using any world coordinate system you need, then map that system to a normalized coordinate system. Finally, you map from the normalized system to device-specific coordinates. You can map entire images or portions of images, and you can use clipping functions to make GKS display or clip portions of images that fall outside the areas you specify.

You map from WC space to NDC space (that is, from the world window to the world viewport) by selecting a normalization transformation. GKS maintains a list of up to 256 normalization transformations.

You map between NDC space and DC space (or in other words, between the workstation window and the workstation viewport), by defining a workstation transformation. You set up a workstation transformation by specifying a workstation window and a workstation viewport. When the handler performs the transformation it replaces the "current transformation" values with the "requested transformation" data, and updates the output display.

2.1.3 Input

Graphic input can come from a variety of devices, such as a keyboard, a mouse, or a light pen.

GKS has several logical input device types. They allow you to enter a single point or a series of points in a stroke, choose from a menu, enter text, or select a value in a range. If you write a full handler, you must provide functions to implement each logical input device type mode you want.

If you write a device handler, you need only write one routine to simulate all input types. The workstation manager uses this one routine to gather data and simulate the other logical input device types.

For input routines that return points, the workstation handler gets input in device coordinates and converts it to NDC using the workstation transformation. For device handler systems, the workstation manager converts the points to NDC, and then passes the NDC coordinates to the kernel.

2.1.4 Output

GKS generates graphic images and sends them to output devices. You can send output to such devices as a terminal screen, a printer, a plotter, or a metafile.

Output is made up of primitives. Primitives are graphic elements such as lines, text, and markers. The appearance of each primitive is controlled by attributes, such as the width, style, and color of lines. When you design your handler you create bundles of attributes to define output types. For example, you can define linetypes with a particular line style, width, and color. At run time, a GKS application can select a bundled output type, as well as set individual attributes. Then it can set bits in an Attribute Source Flag (ASF) to tell GKS whether you want to use the bundled or individual attributes when it draws the primitive.

Both the bundled and individual attributes for each primitive are stored in the GKS state list. When an application calls an output function, the GKS kernel binds the primitive to the current attributes, and passes the attributes to the workstation along with the primitive information.

When the kernel calls an output function, the handler maps the picture from world coordinates to normalized device coordinates using the current normalization transformation. The current transformation is the last one you selected.

Then the handler maps the points to device coordinates using the current workstation transformation, and displays the output. If the workstation cannot perform dynamic regeneration, the output may require a workstation regeneration.

When you write a handler, you must decide how to manage operations such as drawing wider lines and filling areas of the display. If your workstation can draw variable-width lines and fill areas of the display surface, your handler should use these capabilities.

If your device cannot perform operations such as these, you may choose to simulate them in a device handler. If you do, the workstation manager will simulate the operation and pass it to your handler as commands your handler can perform. For example, if your device cannot draw variable-width lines, the workstation manager can simulate thicker lines by commanding your handler to draw several thin lines close together.

2.2 Communication Between the Kernel and Your Handler

The kernel calls your handler routines as functions, passing them all the information they need as formal parameters. Your functions can only return information to the handler in formal parameters and a return status. Parameters for each function are listed in the function description.

Function descriptions also contain status codes. Your function can either return one of the status codes listed in the function descriptions, or another status code specific to your system.

If the function returns one of the status codes in the function description, the GKS kernel responds to the status code automatically. The response may be to proceed (for example, if the code indicates normal successful completion), or it may handle the error using the GKS error-handling procedure (as described in the *DEC GKS Reference Manual*).

If the function returns a status code that is not in the function description, the GKS kernel passes a message to the user. You can develop messages specifically for your handler using the VMS Message utility. For information about developing and implementing messages, see the *VAX/VMS Message Utility Reference Manual*. If you do not implement your own error messages, GKS treats the condition as an unexpected error. The default response to such

an error is a GKS "unexpected errror" warning, followed by any error message the handler generates.

Device handler systems also implicitly pass information to the kernel through the workstation manager through the Workstation Description Table.

2.2.1 DEC GKS Data Types

DEC GKS uses the following data types, as shown in Table 2-3.

Table 2–3: DEC GKS Data Types

Data Type	Description
Integers	32-bit signed integers.
Reals	32-bit F_FLOATING.
Text	Seven- or eight-bit characters
One-dimensional Arrays	Groups of integers, reals, or both. Arrays start at the default minimum index for your programming language (for example, 0 in C, and 1 in FORTRAN or PL/I). Indexes into the array are passed assuming the minimum index is 1, so if you program in a language with a different minimum index, your function must compensate.
Two-dimensional Arrays	Groups of integers, reals, or both. If your programming language does not support two-dimensional arrays, your handler must treat the array as one-dimensional, of the size [NUMBER_ROWS times NUMBER_COLUMNS]. In this case your handler must convert indexes expressed as [ROW,COLUMN] to a one-dimensional index. The kernel either passes indexes into the array in either column-major or row-major format according to a variable in the Workstation Description Table, or passes a flag indicating which format the array is in. In the last case, the handler must interpret the indexes.

2.2.2 Parameter Passing

DEC GKS passes the data types as shown in Table 2-4.

Table 2–4: Passing Mechanisms for DEC GKS Data Types

Data Type	Passed by
Integers	Reference.
Reals	Reference.
Read-only Text Strings	Scalar string descriptor (DSC$K_CLASS_S).
Read/Write Text Strings	Two modifiable arguments passed by reference. The first argument is an integer. When passed to the function it contains the size of a string buffer. When returned from the function it should contain the bytes written to the buffer. The second argument is the string buffer, passed by reference (in other words, a pointer to the buffer).
Arrays	Passed by reference. For a two-dimensional array, the kernel passes these three additional parameters: • NUMBER_COLUMNS • NUMBER_ROWS • COLUMN_MAJOR_FLAG

Chapter 3

Building a Workstation Handler System

This chapter explains how to build a workstation handler system. Chapters 4 through 10 describe the functions you must build into your workstation handler.

At a minimum, your functions must perform the operation described in each function description. However, the method you use to perform each function depends on your device.

You should design your workstation handler to suit the requirements and capabilities of the device you want to support. If your device has built-in graphics capabilities, such as the ability to draw polygon fill areas or to perform clipping, you do not need to perform these tasks in software. If your hardware does not have certain capabilities, you must provide them in your handler functions.

Most workstation handler functions perform a GKS operation that users or application programs request, using a GKS$ command. For example, when the user gives the GKS$POLYLINE command, the kernel runs your polyline function. Note that the function descriptions are written for users who already know what the functions should do. If you are not familiar with GKS functions, or if you want more information about particular functions, see the *DEC GKS Reference Manual*.

3.1 Required Capabilities

In order for your handler to comply with the GKS level 2c implementation standard, you must provide the support listed in Table 3-1. These are the minimum required capabilities. Your system may include additional support.

Items marked with an asterisk (*) state the minimum number of styles required for a primitive. The actual styles are specified in the GKS standard.

Table 3–1: GKS Level 2c Required Capabilities

Item	Minimum Requirement
Foreground colors	1
Linetypes	4*
Linewidths	1
Predefined polyline bundles	5
Settable polyline bundles	20
Marker types	5*
Marker sizes	1
Predefined polymarker bundles	5
Settable polymarker bundles	20
Character- and string-precision text character heights	1
Character- and string-precision text character expansion factors	1
String precision fonts	1
Character precision fonts	1
Stroke precision fonts	2
Predefined text bundles	6
Settable text bundles	20
Predefined patterns (for workstations supporting pattern interior style)	1
Settable patterns (for workstations supporting pattern interior style)	10
Hatch styles (for workstations supporting hatch interior style)	3
Predefined fill area bundles	5
Settable fill area bundles	10
Settable normalization transformations	10

Table 3–1 (Cont.): GKS Level 2c Required Capabilities

Item	Minimum Requirement
Segment priorities (for workstations supporting segment priorities)	2
Input classes	6
Prompt and Echo Types per device	1
Maximum string buffer size (characters)	72
Maximum stroke buffer size (points)	64

3.2 Hardcopy Output Devices

If you are building a handler for a hardcopy device, it should generate output whenever the workstation is cleared. It is important to note that when segments are redrawn, the screen is first cleared, so hardcopy output should be generated. The following functions may cause the screen to be cleared.

If the workstation supports segments, then the workstation handler must determine when to generate hardcopy output. The following functions should generate output:

- Update Workstation—Only if new frame is necessary, and segments are actually redrawn.
- Close Workstation—Should contain an implicit Update Workstation and Clear Workstation.
- Redraw All Segments on Workstation—Always.
- Clear Workstation—Always.

In addition, you can also produce hard copy as a part of any other function.

If the kernel is simulating segments, the workstation should produce hard copy whenever the function Clear Workstation is called, before the workstation is actually cleared. It should also produce hard copy during the Close Workstation function, as a result of an implicit Clear Workstation function call.

3.3 Workstation Handler Data Structures

Your handler must be able to reply to inquiries from the kernel. It must also store data passed from the kernel, and refer to that information when it performs subsequent operations.

Logically, all the information the workstation needs to manage can be divided into two groups. One group contains information that never changes. This information includes your workstation's capabilities and your predefined output primitive representations (or bundles). The other group contains information about the state of your workstation handler at any given time. This information includes the current output bundles, and the current and requested transformations.

The GKS standard refers to the data structure that holds constant information as the Workstation Description Table, or WDT. The data structure containing dynamic information about your workstation is called the Workstation State List, or WSL.

You are not required to provide either data structure. The kernel never reads from or writes directly to any data structure maintained by the workstation handler. This gives you freedom to optimize the data structures which your handler maintains. For example, if your device has a hardware color table, and you can read the hardware color table to implement an Inquire Color Representation function, there is no need to keep a color table in software. You are only required to maintain the information usually associated with the data structures, and be able to return the information when the kernel requests it. However, since you will almost certainly need some form of each of these data structures, the following sections fully describe the Workstation Description Table and the Workstation State List. You should use these sections only as guidelines, and modify them according to your own needs.

3.3.1 Building the Workstation Description Table

The WDT contains information about your workstation's capabilities and pointers to the structures you have built. It is a read-only structure.

The tables in this section list WDT information according to the workstation type. This information is specified by the GKS standard. Your handler must be able to give the kernel all the information listed for the type of workstation you are supporting.

It is possible to support several different workstations with the same handler. If you support several devices that operate similarly, you may choose to build a single handler, and include code for dealing with minor differences between your devices. In this case, you must write a separate WDT for each device.

Several of the items in the WDT can be data structures. The contents of each data structure are described in the following table. Note that you may choose to build the data structure within your WDT, or your WDT may simply contain a pointer to structures located elsewhere.

The data item names in these tables correspond with the data item names in the function descriptions.

Table 3–2 contains a list of items that are required for all workstations.

Table 3–2: Workstation Handler Workstation Description Table Structure

Item	Data Type	Description
WORKSTATION_CATEGORY	Integer	The workstation category. One of: • GKS$K_WSCAT_OUTPUT (0) for OUTPUT • GKS$K_WSCAT_INPUT (1) for INPUT • GKS$K_WSCAT_OUTIN (2) for OUTIN • GKS$K_WSCAT_MO (4) for METAFILE OUTPUT • GKS$K_WSCAT_MI (5) for METAFILE INPUT

Table 3–3 contains a list of items required for all workstation types except MI and MO.

Table 3–3: WDT Items for All Workstation Types Except MI and MO

Item	Data Type	Description
DEV_COORDINATE_UNITS	Integer	The coordinate system the device uses. One of: • GKS$K_METERS (0) for METERS • GKS$K_OTHER_UNITS (1) for OTHER UNITS
DEV_DISPLAY_SPACE_SIZE_X	Real	The X dimension of the display size, in logical device coordinates.
DEV_DISPLAY_SPACE_SIZE_Y	Real	The Y dimension of the display size, in logical device coordinates.
RASTER_DISPLAY_SPACE_SIZE_X	Integer	The X dimension of the display size, in raster units. For raster devices, this is the number of columns in the display. For vector devices, this is the highest resolution possible in the X direction.
RASTER_DISPLAY_SPACE_SIZE_Y	Integer	The Y dimension of the display size, in raster units. For raster devices, this is the number of rows in the display. For vector devices, this is the highest resolution possible in the Y direction.

Table 3–4 contains a list of items you must provide for all OUTPUT or OUTIN workstations.

Table 3–4: WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
DISPLAY_TYPE	Integer	The display classification. One of: • GKS$K_WSCLASS_VECTOR (0) for VECTOR • GKS$K_WSCLASS_RASTER (1) for RASTER • GKS$K_WSCLASS_OTHER (2) for OTHER

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
DMAF_POLYLINE	Integer	A flag indicating whether the device can display a change to the polyline representations without redrawing the entire display. Either: • GKS$K_IMM (0) if the device can display the change without redrawing the entire display. • GKS$K_IRG (1) if the device must redraw the display.
DMAF_POLYMARKER	Integer	A flag indicating whether the device can display a change to the polymarker representations without redrawing the entire display. Either: • GKS$K_IMM (0) if the device can display the change without redrawing the entire display. • GKS$K_IRG (1) if the device must redraw the display.
DMAF_TEXT	Integer	A flag indicating whether the device can display a change to the text representations without redrawing the entire display. Either: • GKS$K_IMM (0) if the device can display the change without redrawing the entire display. • GKS$K_IRG (1) if the device must redraw the display.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
DMAF_FILL	Integer	A flag indicating whether the device can display a change to the fill area representations without redrawing the entire display. Either: • GKS$K_IMM (0) if the device can display the change without redrawing the entire display. • GKS$K_IRG (1) if the device must redraw the display.
DMAF_PATTERN	Integer	A flag indicating whether the device can display a change to the pattern representations without redrawing the entire display. Either: • GKS$K_IMM (0) if the device can display the change without redrawing the entire display. • GKS$K_IRG (1) if the device must redraw the display.
DMAF_COLOR	Integer	A flag indicating whether the device can display a change to the color representations without redrawing the entire display. Either: • GKS$K_IMM (0) if the device can display the change without redrawing the entire display. • GKS$K_IRG (1) if the device must redraw the display.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
DMAF_WS_ TRANSFORMATION	Integer	A flag indicating whether your device can change the workstation transformation without redrawing the entire display. Either: • GKS$K_IMM (0) if the device can display the change without redrawing the entire display. • GKS$K_IRG (1) if the device must redraw the display.
DEF_DEFER_MODE	Integer	The default deferral mode. One of: • GKS$K_ASAP (0) • GKS$K_BNIG (1) • GKS$K_BNIL (2) • GKS$K_ASTI (3)
REGEN_MODE	Integer	The default implicit regeneration mode. This flag controls whether the device will redraw the display immediately upon receiving a change to an attribute that cannot be displayed without regenerating the display. One of: • GKS$K_IRG_SUPPRESSED (0) • GKS$K_IRG_ALLOWED (1) If implicit regeneration is allowed, the device updates the display upon receipt of the change. If implicit regeneration is suppressed, the device updates the display at the next workstation update, or when the regeneration mode is changed.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
NUM_LINE_TYPES	Integer	The number of linetypes the device supports. The GKS standard requires that your device support the following four linetypes: • Solid • Dot • Dashed • Dot-dashed Therefore, the minimum number of linetypes is four. You may also supply additional linetypes, so this value should equal four plus the number of additional linetypes you supply.
LIST_LINE_TYPES	Data structure	The linetypes data structure. This is an integer structure containing the linetypes your device supports. You must list the linetypes required by the GKS standard. You can also support as many additional linetypes as you want. These additional types should be identified with integers less than zero.
NUM_LINEWIDTHS	Integer	The number of linewidths the device supports. One of: • 0 if the device supports a continuous range of widths. • The number of discrete widths your device supports. This must be at least one.
NOMINAL_LINEWIDTH	Real	The width in LDC that the device draws when the line width scale factor is set to 1.0. This must be greater than 0.
MINIMUM_LINEWIDTH	Real	The width in LDC of the narrowest line your device can draw. This must be greater than 0.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
MAXIMUM_LINEWIDTH	Real	The width in LDC of the thickest line your device can draw. This must be greater than or equal to the minimum line width.
NUMBER_PREDEF_PLINE_IND	Integer	The number of predefined polyline bundles.
PREDEF_PLINE_BUNDLES	Data structure	The polyline structure. You must supply at least five bundles, and they must be numbered consecutively from 1. The structure must contain the following information for each bundle: 1. LINE_TYPE— the linetype, integer. 2. LINEWIDTH_SCALE_FACTOR—the linewidth scale factor expressed as a real number. 3. COLOR_INDEX—the index into the device's color table, integer.
NUM_MARKERTYPES	Integer	The number of marker types your handler supports. The GKS standard requires that your device support at least the following five marker types: • Dot • Plus Sign • Asterisk • Diagonal Cross • Circle You may also supply additional marker types, so this value should equal five plus the number of additional marker types you supply.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
LIST_MARKERTYPES	Data structure	The marker type structure. This is an integer structure containing the marker types your device supports. You must list the marker types required by the GKS standard. You can also support as many additional marker types as you want. These additional types should be identified with integers less than zero.
NUM_MSIZES	Integer	The number of marker sizes your device can draw. A zero indicates that the device supports a continuous range.
NOMINAL_MSIZE	Real	The size in LDC of the marker that your device draws when marker size scale factor is set to 1.0. This must be greater than 0.
MINIMUM_MSIZE	Real	The width in LDC of the smallest marker your device can draw. This must be greater than 0.
MAXIMUM_MSIZE	Real	The width in LDC of the largest marker your device can draw. This must be greater than or equal to the minimum marker size.
NUMBER_PREDEF_ PMARK_IND	Integer	The number of predefined poly-marker bundles.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
PREDEF_PMARK_BUNDLES	Data structure	The polymarker bundle structure. You must supply at least five bundles, and they must be numbered consecutively from 1. The structure must contain the following information for each bundle: 1. MARKER_TYPE— the polymarker type. Integer. 2. MSIZE_SCALE_FACTOR— the marker size scale factor expressed as a real number. 3. COLOR_INDEX—an integer index into the device's color table.
NUM_FONT_PREC_PAIRS	Integer	The number of font/precision pairs the device supports. According to the GKS standard, you must supply at least four pairs. Of these, one must be Font 1 with string precision, one must be Font 1 with character precision, and two must be stroke precision.
LIST_FONT_PREC_PAIRS	Data structure	The font information structure. The following information is required for each font: • The font number. Integer. • The font's precision. Integer. Font number 1 must conform to the ASCII character set (defined in ANSI standard x3.4-1977). Implementation-specific fonts must be numbered with a negative integer.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
NUM_CHAR_EXP_ FACTORS	Integer	The number of character expansion factors in Font 1. Either: • 0 if the device supports a continuous range. • The number of discrete expansions your device supports. This must be 1 or greater. Note that for this and the five following items, you only need to include information for Font 1, because this is the only font for which the workstation handler returns information.
MINIMUM_CHAR_EXP_ FACTOR	Real	The minimum character expansion factor for Font 1. This must be greater than 0.
MAXIMUM_CHAR_EXP_ FACTOR	Real	The maximum character expansion factor for Font 1. This must be greater than or equal to the minimum character expansion factor.
NUM_CHAR_HEIGHTS	Integer	The number of character heights in Font 1. Either: • 0 if the device supports a continuous range. • The number of discrete heights your device supports. This must be 1 or greater.
MINIMUM_CHAR_HEIGHT	Real	The minimum character height in Font 1, expressed in LDC. This must be greater than 0.
MAXIMUM_CHAR_HEIGHT	Real	The maximum character height in Font 1, expressed in LDC. This must be greater than or equal to the minimum character height.
NUM_PREDEF_TEXT_IND	Integer	The number of predefined text bundles you will provide.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
PREDEF_TEXT_BUNDLES	Data structure	The text bundle structure. You must supply at least two bundles, and they must be numbered consecutively from 1. The structure must contain the following information for each bundle: 1. FONT— the font, integer. 2. PREC—the text precision expressed as an integer. 3. CHAR_EXP_FACTOR—the character expansion factor expressed as a real number. 4. CHAR_SPACE—the text spacing, expressed as a real number. 5. COLOR_INDEX—an integer index into the device's color table.
NUM_FILL_INTSTYLE	Integer	The number of fill area interior styles. This must be in the range 1-4.
LIST_FILL_INTSTYLE	Data structure	The fill style data structure.
NUM_HATCH_STYLE	Integer	The number of hatch styles available on your device. This must be 0 or greater.
LIST_HATCH_STYLE	Data structure	The hatch style data structure.
NUM_PREDEF_FILL_IND	Integer	The number of predefined fill area style bundles.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
PREDEF_FILL_BUNDLES	Data structure	The fill area bundle structure. You must supply at least five bundles, and they must be numbered consecutively from 1. The structure must contain the following information for each bundle: 1. FILL_INTSTYLE— the fill area style, integer. 2. FILL_STYLE_IND—integer.[1] 3. COLOR_INDEX—an integer index into the device's color table.
NUM_PREDEF_PATT_IND	Integer	The number of predefined pattern representations.
PREDEF_PATT_REPS	Data structure	The pattern bundle structure. This is optional, but if you provide pattern bundles, the structures must be numbered consecutively from 1. The structure must contain the following information for each bundle: 1. PATT_DIM_X—Integer count of the X values in the pattern. 2. PATT_DIM_Y—Integer count of the Y values in the pattern. 3. PATT_ARRAY—The two-dimensional pattern structure.

[1]The meaning of FILL_STYLE_IND depends on the value of FILL_INTSTYLE. If FILL_INTSTYLE is GKS$K_INTSTYLE_PATTERN (2), FILL_STYLE_IND is an index into the pattern bundle table. If FILL_INTSTYLE is GKS$K_INTSTYLE_HATCH (3), FILL_STYLE_IND is a pointer into the hatch style table. If FILL_INTSTYLE is any other value, FILL_STYLE_IND is not used and should be zero.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
NUM_COLORS	Integer	The number of colors your device supports. Either: • 0 if the device supports a continuous range of colors. • The number of discrete colors your device supports. This must be 2 or greater.
COLOR_AVAILABLE	Integer	A flag indicating whether your device has a color display. Either: • GKS$K_MONOCHROME (0) if the device is monochrome. • GKS$K_COLOR (1) if the device has a color display.
NUM_PREDEF_COLOR_REP	Integer	The number of predefined color representations. Must be 2 or greater.
PREDEF_COLOR_REPS	Data structure	The color bundle structure. You must supply at least two bundles, and they must be numbered consecutively from 1. The structure must contain the following information for each bundle: 1. RED—a real number representing the red intensity. 2. GREEN—a real number representing the green intensity. 3. BLUE—a real number representing the blue intensity. The intensity value for each color must be in the range 0 to 1.0 inclusive.
NUM_GDP	Integer	The number of GDPs your device supports. This must be 0 or greater.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
LIST_GDP	Data structure	The GDPstructure. Here you indicate the attributes used for each GDP. You must supply the following information for each GDP: 1. GDP_ID—Integer. The identifier for the primitive. 2. NUM_ATTRIB_USED—Integer. The number of sets of attributes used. 3. LINE_USED—An integer flag. One of: • 0 if the GDP does not use line attributes. • 1 if the primitive uses line attributes. 4. MARK_USED—An integer flag. One of: • 0 if the GDP does not use marker attributes. • 1 if the primitive uses marker attributes. 5. TEXT_USED—An integer flag. One of: • 0 if the GDP does not use text attributes. • 1 if the GDP uses text attributes. 6. FILL_USED—An integer flag. One of: • 0 if the GDP does not use fill attributes. • 1 if the primitive uses text attributes.
MAX_PLINE_BUNDLES	Integer	The maximum number of polyline bundle table entries. Must be 5 or greater.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
MAX_PMARK_BUNDLES	Integer	The maximum number of polymarker bundle table entries. Must be 5 or greater.
MAX_TEXT_BUNDLES	Integer	The maximum number of text bundle table entries. Must be 2 or greater.
MAX_FILL_BUNDLES	Integer	The maximum number of fill area bundle table entries. Must be 5 or greater.
MAX_PATT_IND	Integer	The maximum number of hatch pattern bundle table entries. Must be 0 or greater.
MAX_COLOR_IND	Integer	The maximum number of color intensities. Must be 2 or greater.
NUM_SEGMENT_ PRIORITIES	Integer	The number of segment priorities the device supports. Either: • 0 if the device supports a continuous range of priorities. • The number of priorities. Must be 2 or greater.
DMAF_SEGMENT_XFORM	Integer	A flag indicating whether your device can perform dynamic segment transformations. Either: • GKS$K_IMM (0) for dynamic change. • GKS$K_IRG (1) for no dynamic change.
DMAF_INVISIBILITY	Integer	A flag indicating whether your device can make visible segments invisible dynamically. Either: • GKS$K_IMM (0) for dynamic change. • GKS$K_IRG (1) for no dynamic change.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
DMAF_VISIBILITY	Integer	A flag indicating whether your device can make invisible segments visible dynamically. Either: • GKS$K_IMM (0) for dynamic change. • GKS$K_IRG (1) for no dynamic change.
DMAF_HIGHLIGHTING	Integer	A flag indicating whether your device can perform highlighting dynamically. Either: • GKS$K_IMM (0) for dynamic change. • GKS$K_IRG (1) for no dynamic change.
DMAF_SEGMENT_PRIORITY	Integer	A flag indicating whether your device can change segment priority dynamically. Either: • GKS$K_IMM (0) for dynamic change. • GKS$K_IRG (1) for no dynamic change.
DMAF_SEGMENT_OVERLAP	Integer	A flag indicating whether your device can dynamically update the display while adding primitives to open segments that overlap segments of higher priority. Either: • GKS$K_IMM (0) for dynamic change. • GKS$K_IRG (1) for no dynamic change.

Table 3–4 (Cont.): WDT Items for OUTPUT and OUTIN Workstations

Item	Data Type	Description
DMAF_DELETE_SEGMENT	Integer	A flag indicating whether your device can delete segments dynamically. Either: • GKS$K_IMM (0) for dynamic change. • GKS$K_IRG (1) for no dynamic change.

The rest of the tables in this section list information required by OUTIN or INPUT workstations.

The items in Table 3–5 are required if your workstation supports LOCATOR input. One such structure is needed for each LOCATOR logical input device.

Table 3–5: WDT Items for LOCATOR Logical Input Devices

Item	Data Type	Description
DEVNUM	Integer	The locator device number. Must be 1 or greater.
INIT_LOCN_X	Real	The default initial locator X position in WC.
INIT_LOCN_Y	Real	The default initial locator Y position in WC.
NUM_PROMPT_ ECHO_TYPES	Integer	The number of prompt and echo types supported. Must be 1 or greater.
LIST_PROMPT_ECHO_ TYPES	Structure of integers	List of available prompt and echo types for locator input.
ECHO_AREA	Structure of 4 reals	The default echo area in DC.
LOCATOR_DATA_ RECORD	Structure	The locator data record required for PET 1. This is an implementation-dependent PET, so the data record is also implementation dependent.

The items in Table 3–6 are required if your workstation supports STROKE input. One such structure is needed for each STROKE logical input device.

Table 3–6: WDT Items for STROKE Logical Input Devices

Item	Data Type	Description
DEVNUM	Integer	The stroke device number. Must be 1 or greater.
MAXIMUM_BUFSIZE	Integer	The maximum input buffer size in bytes. Must be 64 or greater.
NUM_PROMPT_ ECHO_TYPES	Integer	The number of prompt and echo types. Must be 1 or greater.
LIST_PROMPT_ECHO_ TYPES	Structure of integers	List of available prompt and echo types for STROKE input.
ECHO_AREA	Structure of 4 reals	The default echo area in LDC.
STROKE_DATA_ RECORD	Structure	The stroke data record required for PET 1. This is an implementation-dependent PET, so the data record is also implementation dependent. It must contain at least the default input buffer size in bytes, expressed as an integer of 1 or greater.

The items in Table 3–7 are required if your workstation supports VALUATOR input. One such structure is required for each VALUATOR logical input device.

Table 3–7: WDT Items for VALUATOR Logical Input Devices

Item	Data Type	Description
DEVNUM	Integer	The valuator device number. Must be 1 or greater.
INIT_VALUE	Real	The default initial value.
NUM_PROMPT_ ECHO_TYPES	Integer	The number of prompt and echo types. Must be 1 or greater.
LIST_PROMPT_ECHO_ TYPES	Structure of integers	List of available prompt and echo types for VALUATOR input.
ECHO_AREA	Structure of 4 reals	The default echo area in LDC.
VALUATOR_DATA_ RECORD	Structure	The valuator data record required for PET 1. This is an implementation-dependent PET, so the data record is also implementation dependent. This must contain at least the default low value and high value, expressed as real numbers.

The items in Table 3–8 are required if your workstation supports CHOICE input. One such structure is needed for each logical input device of type CHOICE.

Table 3–8: WDT Items for CHOICE Logical Input Devices

Item	Data Type	Description
DEVNUM	Integer	The choice device number. Must be 1 or greater.
MAX_NUM_CHOICE	Integer	The maximum number of choices. Must be 1 or greater.
NUM_PROMPT_ ECHO_TYPES	Integer	The number of prompt and echo types. Must be 1 or greater.
LIST_PROMPT_ECHO_ TYPES	Structure of integers	List of available prompt and echo types for CHOICE input.
ECHO_AREA	Structure of 4 reals	The default echo area in LDC.
CHOICE_DATA_ RECORD	Structure	The choice data record required for PET 1. This is an implementation-dependent PET, so the data record is also implementation dependent.

The items in Table 3–9 are required if your workstation supports PICK input. One such structure is required for each PICK logical input device.

Table 3–9: WDT Items for PICK Logical Input Devices

Item	Data Type	Description
DEVNUM	Integer	The pick device number. Must be 1 or greater.
NUM_PROMPT_ ECHO_TYPES	Integer	The number of prompt and echo types. Must be 1 or greater.
LIST_PROMPT_ECHO_ TYPES	Structure of integers	List of available prompt and echo types for PICK input.
ECHO_AREA	Structure of 4 reals	The default echo area in LDC.
PICK_DATA_RECORD	Structure	The pick data record required for PET 1. This is an implementation-dependent PET, so the data record is also implementation dependent.

The items in Table 3–10 are required if your workstation supports STRING input. One such structure is required for each STRING logical input device.

Table 3–10: WDT Items for STRING Logical Input Devices

Item	Data Type	Description
DEVNUM	Integer	The string device number. Must be 1 or greater.
MAXIMUM_BUFSIZE	Integer	The maximum input buffer size in bytes. Must be 72 or greater.
NUM_PROMPT_ ECHO_TYPES	Integer	The number of prompt and echo types. Must be 1 or greater.
LIST_PROMPT_ECHO_ TYPES	Structure of integers	List of available prompt and echo types for STRING input.
ECHO_AREA	Structure of 4 reals	The default echo area in LDC.

Table 3–10 (Cont.): WDT Items for STRING Logical Input Devices

Item	Data Type	Description
STRING_DATA_ RECORD	Structure	The string data record required for PET 1. This is an implementation-dependent PET, so the data record is also implementation dependent. This must contain the default input buffer size and initial cursor position. Both values are integers of 1 or greater.

3.4 Workstation State List

The GKS standard defines the Workstation State List (WSL) as a read/write structure that contains current information about one workstation. You are not required to build a WSL for your workstation handler, but if you want to keep any global read/write information (such as a device channel number), you must store it in the WSL. You might also use the WSL to store primitive representation information and other data.

A separate WSL should exist for each open workstation. The kernel allocates storage for your WSL at open workstation time, and passes the address of that storage space to your Open Workstation function. If you choose to implement a WSL, your Open Workstation function must initialize the WSL at that address. The kernel also passes the address to each function that may need the WSL.

Table 3–11, Table 3–12, and Table 3–13 list the elements of the WSL maintained in the handler. Note that these tables list only suggested items. Your WSL can contain any data you want, in any structure you choose. The only restriction is that your handler must be able to return all of the following information in response to inquiries from the kernel.

The initial value of most entries in the WSL is the same as the corresponding item in the WDT. The following tables list the initial value WDT for these items.

If the initial value shown for an item is something other than WDT, that is the initial value the item should take.

Table 3–11 lists elements that exist for all workstations. Their initial values are passed by the kernel at Open Workstation.

Table 3–11: WSL Elements for All Workstation Types

Item	Data Type
CONNECTION_ID	String
WSTYPE	Integer

Table 3–12 lists elements that exist for OUTIN, OUTPUT, and MO workstations. Note that since the WDT for metafile output (MO) workstations does not contain most of these values, if you are supporting an MO workstation, items marked WDT are actually implementation-dependent constants.

Table 3–12: WSL Elements for OUTIN, OUTPUT, and MO Workstations

Item	Initial Value	Data Type	Description
STORED_ SEGMENTS	Empty	List of integers	The list of stored segments. Not necessary if the kernel is simulating segments.
DEFER_MODE	WDT	Integer	The deferral mode.
REGEN_MODE	WDT	Integer	The regeneration mode. Not necessary if the kernel is simulating segments.
DISPLAY_ EMPTY	GKS$K_EMPTY	Integer	A flag stating whether the display surface is empty.
NEW_FRAME	GKS$K_ NEWFRAME_ NOTNECESSARY	Integer	A flag stating whether a new frame is necessary upon update. Not necessary if the kernel is simulating segments.
GLOBAL_ INTERACTIONS_ PRESENT	GKS$K_FALSE (0)	Integer	A flag stating whether global interactions are in progress. GKS$K_TRUE (1) if there are global interactions.
TRANSFORM_ FLAG	GKS$K_ NOTPENDING (0)	Integer	A flag stating whether a transformation is pending.
NUMBER_ PLINE_IND	WDT	Integer	The number of polyline bundles.
PLINE_ BUNDLES	WDT	Structure	The polyline bundle structure.
NUMBER_ PMARK_IND	WDT	Integer	The number of polymarker bundles.

Table 3–12 (Cont.): WSL Elements for OUTIN, OUTPUT, and MO Workstations

Item	Initial Value	Data Type	Description
PMARK_ BUNDLES	WDT	Structure	The polymarker bundle structure.
NUM_TEXT_ IND	WDT	Integer	The number of text bundles.
TEXT_ BUNDLES	WDT	Structure	The text bundle structure.
NUM_FILL_ IND	WDT	Integer	The number of fill bundles.
FILL_BUNDLES	WDT	Structure	The fill area style bundle structure.
NUM_PATT_ IND	WDT	Integer	The number of pattern bundles.
PATT_ BUNDLES	WDT	Structure	The pattern bundle structure.
NUM_COLOR_ REP	WDT	Integer	The number of color bundles.
COLOR_ BUNDLES	WDT	Integer	The color bundle structure.
WORLD_ WINDOW	0.0, 1.0, 0.0, 1.0	Four reals in WC	The world window.
WORLD_ VIEWPORT	0.0, 1.0, 0.0, 1.0	Four reals in NDC	The world viewport.
REQ_WS_ WINDOW	0.0, 1.0, 0.0, 1.0	Four reals in NDC	The requested world viewport.
REQ_WS_ VIEWPORT	0.0, X_MAX, 0.0, Y_MAX	Four reals in LDC	The requested world viewport. X_MAX and Y_MAX are the X and Y components of the display surface size as listed in the WDT.
CUR_WS_ WINDOW	0.0, 1.0, 0.0, 1.0	Four reals in NDC	The current workstation window.
CUR_WS_ VIEWPORT	0.0, X_MAX, 0.0, Y_MAX	Four reals in LDC	The current workstation viewport. X_MAX and Y_MAX are the X and Y components of the display surface size as listed in the WDT.

For INPUT and OUTIN workstations, the WSL should also include items for each type of logical input device the workstation supports. The following tables list the items for each logical input device type.

Table 3–13: WSL Items for LOCATOR Logical Input Devices

Item	Initial Value	Data Type	Description
DEVNUM	WDT	Integer	The logical input device number.
OPMODE	GKS$K_INPUT_MODE_REQUEST (0)	Integer	The input operating mode.
ECHO_SWITCH	GKS$K_ECHO (1)	Integer	The echo/noecho switch.
XFORM	0	Integer	The normalization transformation the device should use to convert the points to WC.
INIT_LOCN_X, INIT_LOCN_Y	WDT	Real	The initial locator position, in WC.
PROMPT_ECHO_TYPE	1	Integer	The prompt and echo type.
ECHO_AREA	WDT	Array of 4 reals	The initial echo area.
LOC_DATAREC	WDT	Data structure	The locator data record.

Table 3–14: WSL Items for STROKE Logical Input Devices

Item	Initial Value	Data Type	Description
DEVNUM	WDT	Integer	The logical input device number.
OPMODE	GKS$K_ INPUT_MODE_ REQUEST (0)	Integer	The input operating mode.
ECHO_SWITCH	GKS$K_ECHO (1)	Integer	The echo/noecho switch.
XFORM	0	Integer	The normalization transformation the device should use to convert the points to WC.
NUM_INIT_ POINTS	0	Integer	The initial number of points in the input.
INITX_ARRAY, INITY_ARRAY	Empty	Data structure	The initial array of points in the stroke.
PROMPT_ ECHO_TYPE	1	Integer	The prompt and echo type.
ECHO_AREA	WDT	Array of 4 reals	The initial echo area.
STK_DATAREC	WDT	Data structure	The stroke data record, containing at least the input buffer size.

Table 3–15: WSL Items for VALUATOR Logical Input Devices

Item	Initial Value	Data Type	Description
DEVNUM	WDT	Integer	The logical input device number.
OPMODE	GKS$K_ INPUT_MODE_ REQUEST (0)	Integer	The input operating mode.
ECHO_SWITCH	GKS$K_ECHO (1)	Integer	The echo/noecho switch.
INIT_VALUE	WDT	Real	The initial value.
PROMPT_ ECHO_TYPE	1	Integer	The prompt and echo type.
ECHO_AREA	WDT	Array of 4 reals	The initial echo area.
VAL_DATAREC	WDT	Data structure	The valuator data record, containing at least the low and high values.

Table 3–16: WSL Items CHOICE Logical Input Devices

Item	Initial Value	Data Type	Description
DEVNUM	WDT	Integer	The logical input device number.
OPMODE	GKS$K_ INPUT_MODE_ REQUEST (0)	Integer	The input operating mode.
ECHO_SWITCH	GKS$K_ECHO (1)	Integer	The echo/noecho switch.
INIT_STATUS	GKS$K_ NOCHOICE	Integer	The initial choice status.
INIT_CHOICE	Undefined	Integer	The choice that will be returned if the user makes no selection.
PROMPT_ ECHO_TYPE	1	Integer	The prompt and echo type
ECHO_AREA	WDT	Array of 4 reals	The initial echo area.
CHOICE_ DATAREC	WDT	Data structure	The choice data record.

Table 3–17: WSL Items PICK Logical Input Devices

Item	Initial Value	Data Type	Description
DEVNUM	WDT	Integer	The pick logical input device number.
OPMODE	GKS$K_ INPUT_MODE_ REQUEST (0)	Integer	The input operating mode.
ECHO_SWITCH	GKS$K_ECHO (1)	Integer	The echo/noecho switch.
INIT_STATUS	GKS$K_ NOPICK	Integer	The initial pick status.
INIT_SEGMENT	Undefined	Integer	The segment number that will be returned if the user makes no selection.
INITIAL_ PICKID	Undefined	Integer	The pick identifier that will be returned if the user makes no selection.
PROMPT_ ECHO_TYPE	1	Integer	The prompt and echo type.
ECHO_AREA	WDT	Array of 4 reals	The initial echo area.
PICK_ DATAREC	WDT	Data structure	The pick data record.

Table 3–18: WSL Items for STRING Logical Input Devices

Item	Initial Value	Data Type	Description
DEVNUM	WDT	Integer	The string logical input device number.
OPMODE	GKS$K_ INPUT_MODE_ REQUEST (0)	Integer	The input operating mode.
ECHO_SWITCH	GKS$K_ECHO (1)	Integer	The echo/noecho switch.
INIT_STRING	Undefined	Integer	The initial string record.
PROMPT_ ECHO_TYPE	1	Integer	The prompt and echo type.
ECHO_AREA	WDT	Array of 4 reals	The initial echo area.
STRING_ DATAREC	WDT	Data structure	The string data record, containing at least the initial buffer size and initial cursor position.

3.5 Required Functions

After you decide how to manage data for the WDT and WSL, you can develop the functions you need.

Different workstation categories require different functions. The following chapters provide information about each set of functions:

- Chapter 4, Workstation Handler Control and Transformation Functions
- Chapter 5, Workstation Handler Input Functions
- Chapter 6, Workstation Handler Inquiry Functions
- Chapter 7, Workstation Handler Metafile Functions
- Chapter 8, Workstation Handler Set Representation Functions
- Chapter 9, Workstation Handler Output Functions
- Chapter 10, Workstation Handler Segment Functions

Review these chapters to determine which functions you need.

The description of each function includes a discussion of the function's effects. The functions you write should generate the results listed in the function description. Some descriptions provide sample algorithms. In addition, the *DEC GKS Reference Manual* provides a full discussion of each DEC GKS function.

3.5.1 Segment Simulation

You can choose either to support segments within your workstation handler, or to allow the GKS kernel to simulate segments. Simulating segments simplifies your handler and requires fewer functions, but depending on the capabilities of your device, it may be far less efficient.

If you want to support segments within your handler, you must supply the functions described in Chapter 10, Workstation Handler Segment Functions. If you want the GKS kernel to simulate segments, do not provide the functions in Chapter 10, Workstation Handler Segment Functions, but you must supply the functions PERFORM DEFERRED OUTPUT (described in Chapter 4, Workstation Handler Control and Transformation Functions), and HIGHLIGHT EXTENT (described in Chapter 9, Workstation Handler Output Functions).

The kernel determines whether to simulate segments by checking whether segment functions appear in your DFT. If it finds segment functions, it uses them to support segments. If it does not find segment functions, it simulates them.

Note that you must either supply all segment functions, or let the kernel support all segment functions. You cannot support some functions and have the kernel simulate the rest. The macro that builds your DFT checks whether you have included any segment functions, and it will not build your DFT unless you have either included all the segment functions, or none of them. The DFT macro also checks for HIGHLIGHT EXTENT and PERFORM DEFERRED OUTPUT, and will not build your DFT if your handler includes these functions and any segment functions, or if your handler does not include either the segment functions or these two functions.

3.5.2 Message and Constant Files

Your GKS system contains files that define GKS status codes and constants. You should include these files in each of your handler routines if you want to use the predefined codes and constants. The function descriptions in this guide presume that you will use these predefined values.

The status codes are defined a file named GKSMSGS. The constants are defined in GKSDEFS. Both files are stored in the SYS$LIBRARY directory after you install DEC GKS.

Note that there are several versions of each file. There is one version for each language that DEC GKS supports. You should append the file extension for the language you are using to the file names when you include them. Table 3–19 lists the file extension for each language.

Table 3–19: GKSMSGS and GKSDEFS File Extensions

Language	File Extension
Ada®	.ADA
BASIC	.BAS
C	.H
FORTRAN	.FOR
PL/I	.PLI
Pascal	.PAS

®Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

In addition, the file GKSDEFS.PL2 is also provided for handlers written in PL/I. You can use either this file or GKSDEFS.PLI.

3.6 Suggested Escape Function

This section lists the one escape function that the GKS kernel calls other than in response to an application level command. The GKS kernel calls it by calling your handler's Escape function. You should include this function along with any other escape functions that your handler supports.

3.6.1 INQUIRE GDP EXTENT

This escape function should return the extent of GDPs supported by your handler. The function returns the extent in world coordinates. If your handler supplies GDPs not known to the kernel, and you do not provide this function, the GKS kernel sets the extent of unknown GDPs equal to the extent of the world coordinate points passed in the GDP data record.

The escape function identifier for INQUIRE GDP EXTENT is -404, or the constant GKS$K_ESC_INQ_GDP_EXTENT. You should include this function if your handler supports GDPs not known to the GKS kernel. If you do, your handler's Escape function (described in Chapter 4) should perform this operation.

Data is passed to the escape functions in the input data record. For information about data passing to and from the Escape function, see Chapter 4.

The kernel passes a 16-byte input data record to the function. The record is structured as follows:

- Number of integers in the integer array
- Number of real numbers in the real number array
- Number of strings in the string arrays
- Address of the integer array

There are seven integers in this function's integer array, so the first element in the data record is the number 7. There are no real numbers in the real number array, so the second element in the data record is zero. There are no strings, so the third element in the record is zero, and the string arrays are not included in the data record.

The fourth element is a pointer to an eight-element array of integers. The integers contain the following information in the order shown:

1. The address of the workstation state list.
2. The number of points that define the GDP. This number is also the size of the arrays that the next two integers point to.

3. The address of an array containing the X component of each point in the GDP definition.
4. The address of an array containing the Y component of each point in the GDP definition.
5. The identifier of the GDP.
6. The address of the GDP data record.
7. The address of the current attribute list. See Chapter 9 for more information about the attribute array.

Your function returns the GDP extent in a 20-byte output data record. It is structured as follows:

- Number of integers in the integer array
- Number of real numbers in the real number array
- Number of strings in the string arrays
- Address of the integer array
- Address of the real number array

The integer array should contain an error status code, or zero if the status is success, so the first element in the data record is the number 1. The four corners of the extent are returned in the real number array, so the second element in the data record is 4. There are no strings, so the third element in the record is zero, and the string arrays are not included in the data record.

The fourth element is a pointer to a one-element array of integers. The integer should be the status code.

The fifth element is the address of an array of four real numbers. Your function should write the GDP extent to this location. These should be world coordinate values in the order XMIN, XMAX, YMIN, YMAX.

3.7 Developing Your Device Function Table

The Device Function Table (DFT) lists the address of each function you supply. When the kernel needs to run a specific function it finds the address in this table.

To build your DFT, you create a macro that invokes up to four other macros, depending on the functions in your Workstation Handler. These four macros are supplied by DIGITAL.

The following is a list of the different macros in the order in which they must appear in your macro.

1. DFT—Holds header information and the major output and segment functions. This is a required macro.
2. DFT_INPUT—Holds input functions. This is an optional macro.
3. DFT_GKS_INQ—Holds workstation state list inquiry functions.
4. DFT_WS_INQ—Holds workstation description table inquiry functions. This is a required macro.

Your macro must call the macros DFT, DFT_WS_INQ, and DFT_GKS_INQ. Include DFT_INPUT only if your handler is type OUTIN or INPUT.

Table 3–20 through Table 3–23 list the function name and the keyword for each of the macros. Note that items marked with an asterisk must appear in handlers that support segments. If you include any of the 12 functions marked with an asterisk, you must include them all. If you do not, your DFT will not build. Items marked with a dagger must be included in handlers that do not support segments. If you want the kernel to simulate segments for your handler, put these two functions in. You must include both, or your handler will not build. Finally, be sure that you either include the 12 segment functions, *or* the two segment simulation functions. If you include any combination of asterisked and double asterisked functions, your handler will not build.

Table 3–20: Keywords for DFT Macro

Function	Keyword
Handler DFT name	HANDLER
Workstation Category	WS_CAT; one of OUT, IN, OI, MO, MI
Workstation Level	WS_LEVEL; must be 2c
Open Workstation	OPEN_WS
Close Workstation	CLOSE_WS
Clear Workstation	CLEAR_WS
Update Workstation	UPDATE_WS *
Perform Deferred Actions	PERFORM_DEFERRED **
Set Deferral State	SET_DEFER
Set Global Interactions	SET_GLOBAL
Escape	ESC
Create Segment	CREATE_SEG *

Table 3–20 (Cont.): Keywords for DFT Macro

Function	Keyword
Close Segment	CLOSE_SEG *
Rename Segment	RENAME_SEG *
Delete Segment	DELETE_SEG *
Redraw Segment	REDRAW_SEG *
Highlight Extent	HIGHLIGHT_EXT **
Set NDC Transformation	SET_NDC_XFORM
Message	MSG
Polyline	PLINE
Polymarker	PMARKER
Text	TXT
Fill Area	FILL_AREA
Cell Array	CELL_ARRAY
Generalized Drawing Primitive	GDP
Set Polyline Representation	SET_PLINE_REP
Set Polymarker Representation	SET_PMARK_REP
Set Text Representation	SET_TEXT_REP
Set Fill Area Representation	SET_FILL_REP
Set Pattern Representation	SET_PATT_REP
Set Color Representation	SET_COLOR_REP
Set Segment Transformation	SET_SEG_XFORM *
Set Segment Visibility	SET_SEG_VIS *
Set Segment Priority	SET_SEG_PRIOR *
Set Segment Detectability	SET_SEG_DETECT *
Set Segment Highlighting	SET_SEG_HIGH *
Set Workstation Window	SET_WS_WIND
Set Workstation Viewport	SET_WS_VIEW
Set Normalization Transformation	SET_NORM_XFORM

Table 3–21: Keywords for DFT_INPUT Macro

Function	Keyword
Initialize Locator	INIT_LOCATOR
Initialize Stroke	INIT_STROKE
Initialize Valuator	INIT_VALUATOR
Initialize Choice	INIT_CHOICE
Initialize String	INIT_STRING
Initialize Pick	INIT_PICK
Set Locator	SET_LOCATOR
Set Stroke	SET_STROKE
Set Valuator	SET_VALUATOR
Set Choice	SET_CHOICE
Set String	SET_STRING
Set Pick	SET_PICK
Request Locator	REQ_LOCATOR
Request Stroke	REQ_STROKE
Request Valuator	REQ_VALUATOR
Request Choice	REQ_CHOICE
Request String	REQ_STRING
Request Pick	REQ_PICK
Sample Locator	SAM_LOCATOR
Sample Stroke	SAM_STROKE
Sample Valuator	SAM_VALUATOR
Sample Choice	SAM_CHOICE
Sample String	SAM_STRING
Sample Pick	SAM_PICK
Write Item to Metafile	WRITE_ITEM
Get Item Type	GET_ITEM_TYPE
Read Item from Metafile	READ_ITEM

Table 3–22: Keywords for DFT_GKS_INQ Macro

Function	Keyword
Inquire WS Deferral	INQ_WS_DEFER
Inquire List Polyline Representation	INQ_LIST_PLINE
Inquire Polyline Representation	INQ_PLINE_REP
Inquire List Polymark Representation	INQ_LIST_PMARK
Inquire Polymarker Representation	INQ_PMARK_REP
Inquire List Text Representation	INQ_LIST_TEXT
Inquire Text Representation	INQ_TEXT_REP
Inquire Text Extent	INQ_TEXT_EXTENT
Inquire List Fill Area Representation	INQ_LIST_FILL
Inquire Fill Area Representation	INQ_FILL_REP
Inquire List Pattern Representation	INQ_LIST_PATT
Inquire Pattern Representation	INQ_PATT_REP
Inquire List Color Representation	INQ_LIST_COLOR
Inquire Color Representation	INQ_COLOR_REP
Inquire WS Transformation	INQ_WS_XFORM
Inquire Locator State	INQ_LOCATOR_STATE
Inquire Stroke State	INQ_STROKE_STATE
Inquire Valuator State	INQ_VALUATOR_STATE
Inquire Choice State	INQ_CHOICE_STATE
Inquire String State	INQ_STRING_STATE
Inquire Pick State	INQ_PICK_STATE
Inquire Pixel	INQ_PIXEL
Inquire Pixel Dimension	INQ_PIXEL_DIMEN
Inquire Pixel Array	INQ_PIXEL_ARRAY

Table 3–23: Keywords for DFT_WS_INQ Macro

Function	Keyword
Inquire WS Category	INQ_WS_CAT
Inquire WS Classification	INQ_WS_CLASS
Inquire Maximum Display	INQ_MAX_DISP
Inquire Polyline Facilities	INQ_PLINE_FAC
Inquire Predefined Polyline	INQ_PREDEF_PLINE
Inquire Polymark Facilities	INQ_PMARK_FAC
Inquire Predefined Polymark	INQ_PREDEF_PMARK
Inquire Text Facilities	INQ_TEXT_FAC
Inquire Predefined Text	INQ_PREDEF_TEXT
Inquire Fill Area Facilities	INQ_FILL_FAC
Inquire Predefined Fill Area	INQ_PREDEF_FILL
Inquire Pattern Facilities	INQ_PATT_FAC
Inquire Predefined Pattern	INQ_PREDEF_PATT
Inquire Color Facilities	INQ_COLOR_FAC
Inquire Predefined Color	INQ_PREDEF_COLOR
Inquire Available GDPs	INQ_AVAIL_GDP
Inquire GDP	INQ_GDP
Inquire Dynamic Modification of WS Attributes	INQ_DYN_MOD_WS
Inquire Default Deferral	INQ_DFLT_DEFER
Inquire Maximum Length State Tables	INQ_MAX_LEN_STATE_TABLE
Inquire Number Segment Priorities	INQ_NUM_SEG_PRIOR *
Inquire Dynamic Modification of Segment Attributes	INQ_DYN_SEG_ATTR
Inquire Segment Names	INQ_SEG_NAMES
Inquire Number Logical Input Types	INQ_NUM_LOG_INPUT
Inquire Default Locator	INQ_DEF_LOCATOR
Inquire Default Stroke	INQ_DEF_STROKE
Inquire Default Valuator	INQ_DEF_VALUATOR

Table 3–23 (Cont.): Keywords for DFT_WS_INQ Macro

Function	Keyword
Inquire Default Choice	INQ_DEF_CHOICE
Inquire Default String	INQ_DEF_STRING
Inquire Default Pick	INQ_DEF_PICK
Inquire Storage Size	INQ_STORAGE_SIZE

Your macro must contain the following elements:

- A call to the function DFT
- The name of your DFT
- The workstation category
- The GKS implementation level
- Your function names paired with the keywords listed in Table 3–20
- A call DFT_INPUT (if required), followed by the function names paired with the keywords listed in Table 3–21
- A call DFT_GKS_INQ (if required), followed by the function names paired with the keywords listed in Table 3–22
- A call to DFT_WS_INQ, followed by the function names paired with the keywords listed in Table 3–23

For example, suppose you want to assemble a DFT for an OUTIN workstation. Assuming your handler DFT name is WS_HANDLER_DFT, your macro would look like Example 3–1.

Example 3–1: Sample DFT-Building Macro

```
1   DFT -
2             HANDLER    = WS_HANDLER_DFT,-
3             WS_CAT     = 0I,-
4             WS_LEVEL   = 2C,-
5             OPEN_WS    = WS_OPEN,-
    .
    .
    .
              SET_NORM_XFORM = WS_SET_NORM_XFORM
6   DFT_INPUT -
7             INIT_LOCATOR      = WS_INIT_LOC,-
    .
    .
    .
              READ_ITEM         = WS_READ_META
8   DFT_GKS_INQ -
9             INQ_WS_DEFER      = WS_INQ_DEFER,-
    .
    .
    .
              INQ_PIXEL_ARRAY   = WS_INQ_PIX_ARRAY
10  DFT_WS_INQ -
11            INQ_WS_CAT    = WS_INQ_CAT,-
    .
    .
    .
              INQ_STORAGE_SIZE = WS_INQ_STORAGE
```

❶ The first line of your macro must call DFT.

❷ This line names your DFT. The DFT name you supply following HANDLER = will become a global symbol for your DFT. You will use this symbol when you link your handler to the workstation manager.

❸ This line specifies the workstation category. The workstation category must be one of the following:

- OUT (for OUTPUT)
- OI (for OUTIN)
- IN (for INPUT)

❹ This line specifies the GKS level. For a level 2c handler, use 2C.

❺ This line begins the list of keywords shown in Table 3–20, paired with your function names.

❻ This line calls the macro DFT_INPUT. It is necessary because your handler supports input.

❼ This line begins the list of keywords shown in Table 3–21, paired with your function names.

❽ This line calls the macro DFT_GKS_INQ.

❾ This line begins the list of keywords shown in Table 3–22, paired with your function names.

❿ This line calls the macro DFT_WS_INQ.

⓫ This line begins the list of keywords shown in Table 3–23, paired with your function names.

After you write your macro, assemble it with the VMS command:

```
$ MACRO filename +SYS$LIBRARY:GKS$WS_HAND_DFT_MAC/LIB
```

Filename is the name of your macro. Note that the assembler cannot detect spelling errors or incorrect punctuation (such as hyphens or commas). If you encounter unusual errors when you assemble your macro, check these items first.

3.8 Linking Your Handler to DEC GKS

After you build your DFT you should link your handler into a shareable image. To link your handler, do the following.

1. Use the VMS Librarian Utility to put all the object files, *except the device function table,* into an object library. Use any library name you want. For this example, assume you use the name WS_HANDLER.OLB.
2. Give the DCL command:

```
$ LINK/SHAREABLE=WS_HANDLER -
  /MAP=WS_HANDLER /FULL /CROSS -
  WS_HANDLER_DFT, SYS$INPUT /OPTIONS

WS_HANDLER/LIBRARY
UNIVERSAL = WS_HANDLER_DFT
```

For more information about the VMS Librarian Utility, see the *VMS Librarian Reference Manual.* For more information about the VMS Linker Utility, see the *VMS Linker Reference Manual.*

When you finish these steps your handler is linked to DEC GKS and you can test your system.

3.9 Defining Workstation Handler Logical Names

The final step in making your handler work with DEC GKS is defining logical names.

3.9.1 Handler Logical Names

First, define these logical names:

- GKS$WORKSTATION_nn
- GKS$FUNCTION_TAB_nn

In each of these logical names, *nn* is the workstation type you want to use with your handler. This may be any number that is not already used by another handler. To find out what numbers are already in use, use the DCL command SHOW LOGICAL GKS$LIST_TYPES. This command returns a list of the workstation types in use, as shown in the following example.

```
$ SHOW LOGICAL GKS$LIST_TYPES
   "GKS$LIST_TYPES" = "2" (LNM$SYSTEM_TABLE)
       = "3"
       = "5"
       = "10"
       = "11"
       = "12"
       = "13"
       = "14"
       = "15"
       = "31"
       = "32"
       = "33"
       = "34"
       = "38"
       = "51"
       = "52"
       = "53"
       = "54"
       = "55"
       = "56"
       = "61"
       = "70"
       = "72"
$
```

The number 110 does not appear in this list, so assume you have chosen the workstation type 110.

GKS$WORKSTATION_*nn* is defined to be the name of the handler shareable image.

If your shareable image is not installed, you need to define the name of your shareable image as another logical name, and its definition must be the complete file specification of the shareable image.

For example, suppose you want to test the handler DISK$:[SMITH]WS_HANDLER. This handler is not installed, so you must define a logical name with this file specification as its value. For example, enter the command:

```
$ DEFINE WS_HANDLER DISK$:[SMITH]WS_HANDLER
```

Now you can use the logical name WS_HANDLER as the definition of GKS$WORKSTATION_*nn*. Assume you select the workstation number 110. Enter the command:

```
$ DEFINE GKS$WORKSTATION_110 WS_HANDLER
```

GKS$FUNCTION_TAB_*nn* is defined to be the name of the universal symbol used when linking the workstation handler.

For example, if your workstation function table is WS_HANDLER_DFT, enter the command:

```
$ DEFINE GKS$FUNCTION_TAB_110 WS_HANDLER_DFT
```

Next, add your workstation type to the list of workstation types in GKS$LIST_TYPES. To do this locally, repeat the command SHOW LOGICAL GKS$LIST_TYPES, then redefine the logical name to equal each type listed, as well as your new workstation type. For example:

```
$ DEFINE GKS$LIST_TYPES 2, 3, 5, 10, 11, 12, 13, 14, 15, 31, 32, 33, 34,
38, 51, 52, 53, 54, 55, 56, 61, 70, 72, 110
```

```
$ SHOW LOGICAL GKS$LIST_TYPES
   "GKS$LIST_TYPES" = "2" (LNM$SYSTEM_TABLE)
       = "3"
       = "5"
       = "10"
       = "11"
       = "12"
       = "13"
       = "14"
       = "15"
       = "31"
       = "32"
       = "33"
       = "34"
       = "38"
       = "51"
       = "52"
       = "53"
       = "54"
       = "55"
       = "56"
       = "61"
       = "70"
       = "72"
   "GKS$LIST_TYPES" = "2" (LNM$PROCESS_TABLE)
       = "3"
       = "5"
       = "10"
       = "11"
       = "12"
       = "13"
       = "14"
       = "15"
       = "31"
       = "32"
       = "33"
       = "34"
       = "38"
       = "51"
       = "52"
       = "53"
       = "54"
       = "55"
       = "56"
       = "61"
       = "70"
       = "110"
$
```

While you develop your handler, define its workstation number in your process table. When you install your finished handler, define it in the system table.

3.9.2 Adding Logical Names to GKSTARTUP.COM

Finally, place the shareable image in SYS$LIBRARY, and install the image using the VMS Install utility.

While you test your handler, you may wish to define the logical names described in Section 3.9 only locally. When you are finished testing your handler, these logicals should be added to the file SYS$MANAGER:GKSTARTUP.COM.

3.9.3 Reentrance

Your handler must be totally reentrant. This means that the shareable image cannot have "copy-on-reference" pages. You can tell whether your image has copy-on-reference pages by examining the link map that the system produces when you link your shareable image. (See the *VAX/VMS Linker Reference Manual* for more information.)

If your shareable image has copy-on-reference pages, then you must define the logical name GKS$NON_REENTRANT_nn as TRUE. This will make the system activate a new image of the handler each time the workstation is opened.

Do not define this logical name if your device is reentrant.

Note that FORTRAN handlers that use local variables, C handlers that use the C run-time library, or any handlers that write global variables, are non-reentrant.

Chapter 4

Workstation Handler Control and Transformation Functions

This chapter describes the workstation handler control and transformation functions.

In addition, this chapter includes the function PERFORM DEFERRED OUTPUT. You must provide this function only if you want to use the GKS kernel to simulate segments. If you intend to support segments in your handler, do not include the PERFORM DEFERRED OUTPUT function.

This chapter also describes the active attribute array, a data structure that is passed to the CLOSE WORKSTATION control function, as well as to the INITIALIZE INPUT function and several output functions. You need to know the structure of the array to get data for your functions.

4.1 Active Attribute Array

The active attribute array is a dynamic data structure that lists the current attributes for each output primitive. It contains bundle indexes and a list of geometric and nongeometric attributes. The kernel passes the active attribute array to the handler when it calls the CLOSE WORKSTATION control function, the INITIALIZE INPUT function, and many output functions. Your workstation handler functions must be able to find the data they need from this array.

The active attribute array contains current values for each output attribute, read from the GKS State List. The attributes for a particular primitive change each time the application user calls a SET function for that primitive. For example, the GKS$ interface function GKS$SET_LINE_TYPE changes the LINE_TYPE record in the GKS State List. This establishes a new current line type. Then when the user calls GKS$POLYLINE, the kernel passes the current polyline attributes in the active attribute array.

The active attribute array is a read/write structure. However, the handler must treat it as a read-only structure. *Your functions must never write to this structure.* If the handler must change values in the attribute array (for example, for a simulation routine), then it must create a copy of the array.

The kernel passes the entire array to some functions, and passes portions of the array to other functions. The description of each function that receives information from this array lists the portion of the array that the kernel passes, as shown in Table 4-1.

Table 4–1: Active Attribute Array Structure

Item	Data Type
POLYLINE_INDEX	Integer
LINE_TYPE	Integer
LINEWIDTH_SCALE_FACTOR	F-float
POLYLINE_COLOR_INDEX	Integer
PICK_ID	Integer
POLYMARKER_INDEX	Integer
MARKERTYPE	Integer
MARKSIZE_SCALE_FACTOR	F-float
POLYMARKER_COLOR_INDEX	Integer
PICK_ID	Integer
FILL_AREA_INDEX	Integer
INTERIOR_STYLE	Integer
FILL_STYLE_INDEX	Integer
FILL_AREA_COLOR_INDEX	Integer
PATTERN_REFERENCE_POINT_X	F-float
PATTERN_REFERENCE_POINT_Y	F-float
PATTERN_HEIGHT_X	F-float
PATTERN_HEIGHT_Y	F-float
PATTERN_WIDTH_X	F-float
PATTERN_WIDTH_Y	F-float
PICK_ID	Integer
TEXT_INDEX	Integer

Table 4–1 (Cont.): Active Attribute Array Structure

Item	Data Type
FONT	Integer
PRECISION	Integer
CHAR_EXP_FACTOR	F-float
CHAR_SPACE	F-float
TEXT_COLOR_INDEX	Integer
CHARACTER_HEIGHT_X	F-float
CHARACTER_HEIGHT_Y	F-float
CHARACTER_WIDTH_X	F-float
CHARACTER_WIDTH_Y	F-float
TEXT_PATH	Integer
TEXT_ALIGNMENT_HORIZ	Integer
TEXT_ALIGNMENT_VERT	Integer
PICK_ID	Integer
CELL_ARRAY_PICK_ID	Integer

4.2 Function Descriptions

This section contains the control and transformation function descriptions.

Open Workstation

This function opens the workstation and makes it available for input and output. It must do the following:

- Allocate and initialize the workstation state list, as described in Chapter 3, Building a Workstation Handler System.
- Clear the display surface, if it is not clear.
- Assign a channel to the physical device.
- Open and initialize any input devices.
- Perform any device-specific initialization.

It is possible for one handler to support more than one device type. If your handler does, you can use the WSTYPE value to identify the device type that the kernel is opening, and load the WDT for that device type.

Required

Required for all workstations.

Input Parameters

WSL	The address of the handler's local data area.
WSTYPE	The workstation type expressed as an integer value. The low-order word is the actual workstation type. The high-order word is any device specific information.
DEVNAME	String. The actual device name. Passed by descriptor.
WS_ID	Integer. The workstation identification number.

Output Parameters

LEVEL	Integer. The GKS level the handler supports. In this implementation your handler should return the value GKS$K_LEVEL_2C (8).
CALL_BACK_TABLE	Address. An array of function addresses as specified in Table 4–2.

Table 4–2: Call Back Table Output Parameter

Function	Data Type
GKS$INQUIRE_DEVICE_HAND_WDT()	Integer
GKS$SIM_STROKE_TEXT()	Integer
GKS$SIM_STROKE_TEXT_EXTENT()	Integer
GKS$STORE_EVENTS()	Integer
GKS$INPUT_OUTPUT_CONFLICT()	Integer
GKS$INPUT_REDRAW_INPUT()	Integer
GKS$INPUT_UPDATE_INPUT()	Integer
GKS$INPUT_REFRESH_INPUT()	Integer
GKS$INPUT_SET_WS_VIEWPORT()	Integer
GKS$FIND_SEGMENT()	Integer
GKS$FIND_SEG_EXTENT()	Integer

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_26	Specified workstation cannot be opened.

Close Workstation

This function closes the workstation. It must do the following:

- Do any clean-up required by your device or your handler.
- Clear the display (if desired).
- Deassign the device channel.

If your workstation is type MO, it should use the ATTRIB_ARRAY values to update the values in the metafile. Other workstation types need not use the attribute array.

Required

Required for all workstations.

Input Parameters

WSL	The address of the handler's WSL.
ATTRIB_ARRAY	The active attribute array explained in Section 4.1.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Clear Workstation

This function clears the display surface if necessary. If your handler supports segments, it performs the following operations:

- Flushes out any deferred output.
- Clears the workstation, performing any requested workstation transformations.
- Deletes all segments on this workstation.
- Sets NEW_FRAME to NO.
- Sets DISPLAY_EMPTY to EMPTY.

If your handler does not support segments, it performs the following operations:

- For a hardcopy device, generates hard copy.
- Clears the workstation, performing any requested workstation transformations.
- Sets DISPLAY_EMPTY to EMPTY.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
CLEAR_FLAG	An integer flag that controls whether to clear the display surface. If the flag is GKS$K_CLEAR_ALWAYS (1), then always clear the surface. If the flag is GKS$K_CLEAR_CONDITIONALLY (0), then clear the surface only if it is not already clear.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Update Workstation

This function is only required if the kernel is not simulating segments for this device. The function performs all deferred actions for the workstation, possibly by calling a PERFORM DEFERRED ACTIONS routine, without first clearing the display surface. Then, if the DEF_MODE flag is PERFORM and NEW_FRAME is YES, it performs the following steps in order.

1. If DISPLAY_EMPTY is NOTEMPTY, the function clears the display surface and sets the DISPLAY_EMPTY to EMPTY.
2. If the flag TRANSFORM_FLAG is PENDING, meaning a transformation is pending on the workstation, it assigns the values of REQUESTED_WS_WINDOW to CURRENT_WS_WINDOW, and REQUESTED_WS_VIEWPORT to CURRENT_WS_VIEWPORT. Then it sets TRANSFORM_FLAG to NOTPENDING.

   These flags and values are stored in the WSL. Note that the transformation update may require the handler to recompute geometric attributes.
3. If there are visible segments on the workstation, the function redisplays them, and sets DISPLAY_EMPTY to NOTEMPTY.
4. Sets NEW_FRAME to NO.

Note that these operations are equivalent to the following:

- If NEW_FRAME is YES, or if DEF_MODE is PERFORM, perform the function REDRAW ALL SEGMENTS ON WORKSTATION (described later in this chapter).
- If NEW_FRAME is NO, or if DEF_MODE is POSTPONE, perform all deferred actions.

Conditional

This function is required only if the kernel is not simulating segments for the device.

Update Workstation

Input Parameters

WSL	The address of the handler's local data area.
UPDATE_ REGENERATION_FLAG	Integer. Either GKS$K_POSTPONE_FLAG (0) or GKS$K_PERFORM_FLAG (1).

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Perform Deferred Output

This function flushes any buffered output to the device. This function is also used to simulate deferral modes for devices that cannot change modes, and it is required by all workstations regardless of their implicit deferral mode. That is, if a workstation's deferral mode is ASAP and cannot be changed (such as the metafile output), this function is required even though all it does is return GKS$_SUCCESS.

Conditional

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler does not support segments.

Input Parameters

WSL	The address of the handler's local data area.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Escape

This function performs escape operations specified by the ESCAPE_ID parameter. Note that the kernel does not look at the data records passed to or from this function. The kernel only checks the first longword of the input data record for the WS_ID value, then passes the entire record directly to the handler.

Note that applications can call escape functions for devices that are not open. For example, the "inquire list of escape functions" escape function can be called without the device being open. In this case, the kernel loads the device on which the escape function should be performed, and calls that device's escape routine. When it does, it sets the WSL parameter to zero.

Required

Required for all workstations.

Previous Initialization Required

Some device-dependent initialization may be required.

Input Parameters

WSL	The address of the handler's local data area.
ESCAPE_ID	Integer specifying what escape function this is.
IN_RECORD_SIZE	The size in bytes of the data record array being passed, not including the first longword.
IN_RECORD_ARRAY	The actual data record array. Passed starting at the second longword. The first longword is used by the kernel to obtain the workstation id number.

Modified Parameters

OUT_RECORD_SIZE	The size in bytes of the output data record. On input this is the size of the array. On output the number of bytes written into the array.

Output Parameters

OUT_RECORD_ARRAY	The output data record.
TOTAL_RECORD_SIZE	The total size in bytes of the output data record. This may be longer than the returned size if the output data record was not large enough.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_180	Specified escape function is not supported.
GKS$_ERROR_181	Specified escape function identification is invalid.
GKS$_ERROR_182	Contents of escape data record are invalid.

Set Workstation Window

This function sets the workstation window, according to the following algorithm:

1. Sets REQUESTED_WS_WINDOW according to the input parameter WINDOW_LIMITS.
2. If DYNAMIC_MODIFICATION_ACCEPTED_FOR_WORKSTATION_TRANSFORMATION is IMM, or if DISPLAY_EMPTY is EMPTY:
   - Sets CURRENT_WS_WINDOW to the new window specified in the input parameter WINDOW_LIMITS.
   - Recomputes geometric attributes as needed.
   - Sets TRANSFORM_FLAG to NOTPENDING, and returns TRANSFORM_FLAG = NOTPENDING.
3. Otherwise, if DYNAMIC_MODIFICATION_ACCEPTED_FOR_WORKSTATION_TRANSFORMATION is IRG and DISPLAY_EMPTY is NOTEMPTY:
   - Sets TRANSFORM_FLAG to PENDING.
   - If the handler is performing segments and REGEN_MODE is ALLOWED, calls REDRAW ALL SEGMENTS.
   - If the handler is performing segments and REGEN_MODE is SUPPRESSED, sets NEW_FRAME to YES.
4. Returns the value of TRANSFORM_FLAG as PENDING.

All flags shown in this algorithm are described in Chapter 3. They are normally maintained in the WSL.

This function may change the clipping rectangle and the normalization transformation.

Required

Required for OUTPUT, OUTIN, INPUT, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
WINDOW_LIMITS	The requested workstation window in NDC. Ordered as XMIN, XMAX, YMIN, YMAX.

Output Parameters

TRANSFORM_FLAG	Integer. Either GKS$K_NOTPENDING (0) or GKS$K_PENDING (1).

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set Workstation Viewport

This function sets the workstation viewport, according to the following algorithm:

1. Sets REQUESTED_WS_VIEWPORT according to the input parameter VIEWPORT_LIMITS.
2. If DYNAMIC_MODIFICATION_ACCEPTED_FOR_WORKSTATION_TRANSFORMATION is IMM, or if DISPLAY_EMPTY is EMPTY:
   - Sets CURRENT_WS_VIEWPORT equal to the input parameter VIEWPORT_LIMITS.
   - Recomputes geometric attributes as needed.
   - Sets TRANSFORM_FLAG to NOTPENDING.
   - Returns the value GKS$K_NOTPENDING (0) as PENDING.
3. Otherwise, if DYNAMIC_MODIFICATION_ACCEPTED_FOR_WORKSTATION_TRANSFORMATION transformation is IRG and DISPLAY_EMPTY is NOTEMPTY:
   - Sets TRANSFORM_FLAG to PENDING.
   - If the handler is performing segments and REGEN_MODE is ALLOWED, calls REDRAW ALL SEGMENTS.
   - If the handler is performing segments and REGEN_MODE is SUPPRESSED, sets NEW_FRAME to YES.
4. Returns the value of PENDING as TRANSFORM_FLAG.

All flags shown in this algorithm are described in Chapter 3. They are normally maintained in the WSL.

This function may change the clipping rectangle and the normalization transformation.

Required

Required for OUTPUT, OUTIN, INPUT, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
VIEWPORT_LIMITS	Array of four real numbers defining the workstation viewport in LDC. Ordered as XMIN, XMAX, YMIN, YMAX.

Output Parameters

TRANSFORM_FLAG	Integer. Either GKS$K_NOTPENDING (0) or GKS$K_PENDING (1).

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_54	Workstation viewport is not within the display space.

Set Normalization Transformation

This function establishes a new normalization transformation. The transformation information is usually stored in the WSL.

This function changes the clipping rectangle. Also, the handler may need to recompute geometric attributes.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
WINDOW	Array of four real numbers in WC. The world window limits, passed as a (1 x 4) array, ordered as XMIN, XMAX, YMIN, YMAX.
VIEWPORT	Array of four real numbers in NDC. The world normalization viewport limits, passed as a (1 x 4) array, ordered as XMIN, XMAX, YMIN, YMAX.
CLIP_FLAG	Integer. If GKS$K_CLIP (1), clipping is enabled and the clipping rectangle is the viewport or the workstation window, whichever is smallest. If GKS$K_NOCLIP (0), the clipping rectangle is the workstation window.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set Deferral Mode

This function establishes a new deferral mode and implicit regeneration mode. These values are usually stored in the WSL. If the new implicit regeneration mode is GKS$K_IRG_ALLOWED (1), or the new deferral mode is a higher mode than the one currently in effect, then the function also performs any necessary regeneration.

Input Parameters

WSL
The address of the handler's local data area.

DEF_MODE
Integer value of deferral mode. One of:

- GKS$K_ASAP (0)
- GKS$K_BNIG (1)
- GKS$K_BNIL (2)
- GKS$K_ASTI (3)

REGEN_MODE
Integer value of the regeneration mode. One of:

- GKS$K_IRG_SUPPRESSED (0)
- GKS$K_IRG_ALLOWED (1)

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success

Redraw All Segments on Workstation

This function redraws all segments associated with the workstation. It performs the following operations:

- Executes all deferred actions.
- Clears the display surface if necessary.
- Performs any pending workstation transformation update.
- Redraws all visible segments.
- Sets NEW_FRAME_ACTION to GKS$K_NEWFRAME_NOTNECESSARY (0).
- Sets DISPLAY_EMPTY to GKS$K_NOTEMPTY (0), if necessary.

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set Global Interactions

This function signals whether global interactions are present for the workstation. This information is usually stored in the WSL. The function also performs any actions necessary for deferrals.

This function is needed in order to perform BNIG deferral. If the device is in BNIG deferral mode and GLOBAL_INTERACTIONS_PRESENT is GKS$K_TRUE (1) , then all deferred actions must be performed, and actions may not be deferred as long as the workstation remains in BNIG mode, or until this function is executed with GLOBAL_INTERACTIONS_PRESENT = GKS$K_FALSE (0). This means that as long as GLOBAL_INTERACTIONS_PRESENT is GKS$K_TRUE (1), output operations should be performed as soon as possible.

The workstation handler is responsible for performing its own BNIL deferral. That is, if it is in BNIL deferral mode and an input is in progress on that particular workstation, then that workstation should perform all deferred actions, and it should perform all output operations as soon as possible as long as input is in progress.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSL	The address of the handler's local data area.
GLOBAL_ INTERACTIONS_ PRESENT	Integer. GKS$K_FALSE (0) means no global interactions are in progress. GKS$K_TRUE (1) means global interactions are in progress.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Message

This function sends a message to the workstation. It may write the message on the workstation display or on a separate device associated with the workstation. It may also affect the workstation in a purely local way.

Required

Required for MO, OUTPUT, OUTIN, and INPUT workstations.

Input Parameters

WSL	The address of the handler's local data area.
MESSAGE	String descriptor pointing to the message to be displayed.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set NDC Transformation

This function sets the segment's NDC transformation. See Appendix A, Transformations, for a discussion of the NDC transformation. Note that as a result of this function the handler may need to recompute geometric attributes.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
NDC_TRANSFORM	A 1 x 6 array of real numbers of the transformation ordered as M(1,1), M(1,2), M(1,3), M(2,1), M(2,2), M(2,3). Locations M(1,3) and M(2,3) are in NDC.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Chapter 5

Workstation Handler Input Functions

This chapter describes the workstation handler input functions.

5.1 Writing Input Functions

You only need to write input functions if your device is type OUTIN or INPUT. If it is, you must supply the INITIALIZE, SET, and REQUEST functions for the types of input your workstation supports. The GKS standard requires that OUTIN workstations be able to perform all six input types (that is, CHOICE, LOCATOR, VALUATOR, STRING, STROKE, and PICK input), so in order to support an OUTIN workstation you must provide the INITIALIZE, SET, REQUEST, and SAMPLE functions for all six input types.

Before you write PICK input functions, read Appendix C, Pick Simulation Functions. This appendix describes built-in functions you can use to make PICK functions easier to write.

The GKS standard also states that INPUT workstations can support from one to all six input types, so for an INPUT workstation you must supply the INITIALIZE, SET, REQUEST and SAMPLE functions for at least one input type, and as many input types as your device supports.

5.1.1 INITIALIZE Functions

The INITIALIZE functions prepare your workstation for the input operation. These preparations include identifying the logical input device on which you will do input, and passing the current output attributes, the echo area, and input type-specific data to the workstation.

The kernel passes a data record in each initialize function. The format varies depending on the input type. The data record for each input type is described in each initialize function description.

The workstation should store the data that the kernel passes. Your REQUEST INPUT functions will probably need this information for the input operation.

5.1.2 SET Functions

The SET functions set the workstation's operating mode and echo flag to values passed as input parameters. They must store these values for use by the REQUEST INPUT functions.

5.2 REQUEST, SAMPLE, and EVENT Input

The REQUEST and SAMPLE functions get input from the workstation. Each REQUEST function prepares the workstation to accept input, prompts the user for input, updates the display surface to show the input (if echoing is set to GKS$K_ECHO (1)), and stores the input data. When the user signals that the input is complete (in some workstation-specific manner), the function sends data to the kernel, and passes a "success" status code. If the user invokes a BREAK operation to cancel input, the handler should return the status code "none". At the end of the input operation the function sets the workstation back to output.

The SAMPLE functions return the current value of a logical input device. That is, they do not wait for the user to signal that input is complete; rather, they return whatever the value of the input device is at the moment the function is executed.

EVENT input returns data when a predefined event occurs. GKS does not require separate EVENT input functions.

5.2.1 Managing SAMPLE and EVENT Input

When the kernel calls one of your handler's SET MODE functions (for example, Set Locator Mode) for SAMPLE or EVENT mode input, your function should start a conceptual subprocess that monitors the input device. This can be done in VMS through AST routines, updating the data structures asynchronously when input is performed by the user.

You can use the same subprocess for SAMPLE or EVENT input. The only difference between sample and event mode input from the workstation handler's perspective is that for event mode input, the routine must call the built-in function GKS$STORE_EVENTS to return the input data when a trigger is fired. Section 5.2.2 describes GKS$STORE_EVENTS. For sample mode input, triggers are ignored and the SAMPLE function returns the routine's current values when the kernel calls it.

In cases where more than one logical input device is active on a single workstation, one action may be interpreted as the trigger for more than one input function. For example, a single carriage return can trigger both LOCATOR and STROKE input in request or event mode. In this case, the handler must update all affected logical input devices during the AST routine for the input. For event mode input, if you have a trigger for more than one input device, this information will be passed to the kernel in GKS$STORE_EVENTS.

Also, if your workstation handler is echoing input with complement mode during sample or event mode input, and the echo area overlaps with an output window, it is possible that the handler may not always complement out the cursor. For example, a cursor drawn in the forground color can be overdrawn by output in the background color. When the input function tries to move the cursor by first complementing out the original cursor, it finds the color currently at the cursor's location, and draws in that color's complement. Since the original cursor's location is now filled by the background color, the input function draws in the foreground color, and the result is that the cursor reappears rather than disappears. To solve this problem, your handler can set a flag whenever input and output are intermixed, then erase the input echoing primitives whenever it draws overlapping output, and redraw the input echoing after it draws the output.

Some functions require function-specific processing as well. These requirements are explained in the function descriptions in this chapter.

5.2.2 GKS$STORE_EVENTS

This section describes the built-in function GKS$STORE_EVENTS, which your handler calls to return event-mode input. When your handler calls it, the function checks whether all the events can be added to the event queue. If they can, it adds the events to the queue and returns a success message. If not, it returns an error status and does not add any of the events to the queue. If this happens, the function that called GKS$STORE_EVENTS must report the error to the user.

Input Parameters

GKS$STORE_EVENTS accepts the following input parameters:

WS_ID	Workstation id for the device that is reporting these events. Integer, passed by reference.
NUM_SIM_EVENTS	Number of simultaneous events that are being reported. Integer, passed by reference.
EVENT_DATA	Data structure, passed by reference. This structure contains information about each event being reported. There is one iteration of this structure for each simultaneous event. The structure contains these items: • Log_dev_num_n: the logical device where event N occurred. Integer. • Input_class_n: the input class of event N. Integer. • Data_n: An array containing the data record of event N. Its format depends on the input class.

The value of Input_class_n is one of the following:

- GKS$K_INPUT_CLASS_LOCATOR (1)
- GKS$K_INPUT_CLASS_STROKE (2)
- GKS$K_INPUT_CLASS_VALUATOR (3)
- GKS$K_INPUT_CLASS_CHOICE (4)
- GKS$K_INPUT_CLASS_PICK (5)
- GKS$K_INPUT_CLASS_STRING (6)

The contents of Data_n depend on the input type. For Locator input, it contains these three items:

- pos_x—The X component of a point, expressed in NDC. Real.
- pos_y—The Y component of a point, expressed in NDC. Real.
- fill —Four fill characters. String.

For Stroke input, Data_n contains this information:

- npts —The number of points in the stroke. Integer.
- pos_x—Pointer to an array. The array contains the X component of each point, expressed in NDC.
- pos_y—Pointer to an array. The array contains the Y component of each point, expressed in NDC.

For Choice input, Data_n contains this information:

- status—The choice status, either GKS$K_STATUS_OK or GKS$K_STATUS_NOCHOICE. Integer.
- choice_num—The choice number, meaningful if status equals GKS$K_STATUS_OK. Integer.
- fill —Four fill characters. String.

For Valuator input, Data_n contains this information:

- value—The measure of the valuator device. Real.
- fill —Eight fill characters. String.

For String input, Data_n contains this information:

- length—The size of the string. Integer.
- string—The pointer to the string buffer.
- fill —Four fill characters. String.

For Pick input, Data_n contains this information:

- status—The pick status, either GKS$K_STATUS_OK or GKS$K_STATUS_NOPICK. Integer.
- pick_id—The pick id, meaningful only if status equals GKS$K_STATUS_OK. Integer.
- seg_name—The segment name, meaningful only if status equals GKS$K_STATUS_OK. Integer.

The structure can be expressed as the following:

```
struct report_events_type
    {
    int input_class;   /* input class of the event */
    int     logical_dev_num;
    union {            /* only one of these is 'active' */
        struct report_locator locator; /* at a time. See structure   */
        struct report_stroke  stroke;
        struct report_choice  choice;
        struct report_valuator valuator;
        struct report_string  string;
        struct report_pick   pick;
        } report;
    };
```

Output Parameters

GKS$STORE_EVENTS returns the following output parameter:

ALLOWED	GKS$K_TRUE if the events fit on the event queue, or GKS$K_FALSE if they did not.

5.3 Function Descriptions

This section contains the input function descriptions.

Initialize Locator

This function initializes the device to accept input of type LOCATOR. It should store the input parameters in the WSL for the logical input device you specify.

The GKS standard specifies the following prompt and echo type (PET) definitions for LOCATOR input:

PET	Definition
Less than 0	Prompting and echoing is LOCATOR device dependent.
1	Designate the current locator position using an implementation-defined technique.
2	Designate the current locator position using a vertical line and a horizontal line spanning the display surface or the workstation viewport, and intersecting at the current locator position.
3	Designate the current locator position using a tracking cross.
4	Designate the current locator position using a rubber-band line connecting the initial locator position (given in the input parameters) and the current locator position.
5	Designate the current locator position using a rectangle. The diagonal of the rectangle is the line connecting the initial locator position (given in the input parameters) and the current locator position.
6	Display a digital representation of the current locator position, in locator device-dependent coordinates, within the echo area.
7 or greater	Reserved for registration or future standardization.

The data record depends on the PET. For user-defined PETs (those with numbers less than 0), the data record is user defined.

The kernel passes the data record exactly as specified by the application. The workstation handler must know the format of the data records the application will pass for each PET.

The GKS standard defines data records for PETs 4 and 5. For PET 4, the first item in the data record is the following:

Item	Data Type
Attribute control flag	Integer. One of GKS$K_ACF_CURRENT (0) or GKS$K_ACF_SPECIFIED (1).

The remainder of the data record depends on the setting of Attribute Control Flag. If Attribute Control Flag equals GKS$K_ACF_SPECIFIED (1), the remainder of the data record is as follows:

Item	Data Type
Linetype ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Linewidth Scale Factor ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Polyline Color Index ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Polyline Index	Integer. A defined polyline bundle index.
Linetype Index	Integer. A defined line type.
Linewidth Scale Factor	Real. The linewidth scale factor.
Polyline Color Index	Integer. A defined color bundle index.

If Attribute Control Flag equals GKS$K_ACF_CURRENT (0), the current polyline attributes at LOCATOR initialization are used.

For PET 5, the first two items in the data record are as follows:

Item	Data Type
Polyline/Fill Area Control Flag	Integer. One of: • GKS$K_ACF_POLYLINE (0) • GKS$K_ACF_FILL_AREA (1)
Attribute Control Flag	Integer. One of: • GKS$K_ACF_CURRENT (0) • GKS$K_ACF_SPECIFIED (1)

The remainder of the data record depends on the settings of Polyline and Attribute Control Flag. If Attribute Control Flag equals GKS$K_ACF_SPECIFIED (1) and Polyline/Fill Area equals GKS$K_ACF_POLYLINE (0), the remainder of the data record is as follows:

Item	Data Type
Linetype ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Linewidth Scale Factor ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Polyline Color Index ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)

Item	Data Type
Polyline Index	Integer. A defined polyline bundle index.
Linetype Index	Integer. A defined linetype.
Linewidth Scale Factor	Real. The linewidth scale factor.
Polyline Color Index	Integer. A defined color bundle index.

If Attribute Control Flag equals GKS$K_ACF_SPECIFIED (1) and Polyline/Fill Area is GKS$K_ACF_FILL_AREA (1), then the remainder of the data record is as follows:

Item	Data Type
Fill Area Interior Style ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Fill Area Style Index ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Fill Area Color Index ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Fill Area Index	Integer. A defined fill area bundle index.
Fill Area Interior Style	Integer. A defined fill area interior style.
Fill Area Style Index	Integer. A defined fill area style.
Fill Area Color Index	Integer. A defined color bundle index.

If Attribute Control Flag equals GKS$K_ACF_CURRENT (0) and Polyline/Fill Area is GKS$K_ACF_POLYLINE (0), the current polyline attributes at LOCATOR initialization are used.

If Attribute Control Flag equals GKS$K_ACF_CURRENT (0) and Polyline/Fill Area is GKS$K_ACF_FILL_AREA (1), the current polyline attributes at LOCATOR initialization are used.

Required

Required for OUTIN workstations and for INPUT workstations that support LOCATOR input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer value of the locator device number.
XFORM	Integer. The transformation used to convert the initial position from WC to NDC. It is not used by the handler, but must be stored for return in the INQUIRE LOCATOR DEVICE STATE function.
INIT_LOCN_X	Real. The X value of the initial locator position, in NDC.
INIT_LOCN_Y	Real. The Y value of the initial locator position, in NDC.
PROMPT_ECHO_TYPE	Integer. The desired prompt and echo type.
ECHO_AREA	A [1 x 4] array of real numbers defining the allowable echo array in LDC. Ordered as XMIN, XMAX, YMIN, YMAX.
DATA_REC_SIZE	Integer. Number of bytes in the data record array.
LOC_DATAREC	The locator data record. Passed by reference. The data record is passed straight through from the kernel to the handler.
ATTRIB_ARRAY	An array holding the current values of the entire attribute array. This array is defined in Section 4.1. It is supplied to give the handler the information necessary to perform the various PETs.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_60	Polyline index is invalid.
GKS$_ERROR_63	Linetype is equal to zero.
GKS$_ERROR_65	Linewidth scale factor is less than zero.
GKS$_ERROR_80	Fill area index is invalid.
GKS$_ERROR_84	Pattern or hatch style index is equal to zero.
GKS$_ERROR_92	Color index is less that zero.
GKS$_ERROR_140	Specified input device is not present on this workstation.
GKS$_ERROR_141	Specified input device is not in REQUEST mode.
GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation.
GKS$_ERROR_145	Echo area is outside display space.
GKS$_ERROR_146	Contents of input data record are invalid.
GKS$_ERROR_152	Initial value is invalid.

Initialize Stroke

This function initializes the logical input device to accept input of type STROKE. It should store the input parameters in the WSL entries for the specified stroke device.

The GKS standard specifies the following prompt and echo type definitions for STROKE input:

PET	Definition
Less than 0	Prompting and echoing is stroke device dependent.
1	Display the current stroke using an implementation-defined technique.
2	Display a digital representation of the current stroke position, in device-dependent coordinates, within the echo area.
3	Display a marker at each point of the current stroke.
4	Display a line joining successive points of the current stroke.
5 or greater	Reserved for registration or future standardization.

The data record depends on the PET. For user-defined PETs (those with numbers less than 0), the data record is user-defined.

Data records for all PETs contain the following four items. For PETs 1 and 2, these are the only required items for the data record.

Item	Data Type
Input Buffer Size	Integer. The size of the input buffer.
Editing Position	Integer. The initial editing position.
X, Y Interval	Array of two reals. The distance in NDC between points on the stroke, on both the X and Y axis. The cursor must be moved at least this distance before the handler accepts an input point.
Time Interval	Real. The minimum time allowed between points. This interval must expire before the handler accepts an input point, even though the cursor may have moved the required distance.

For PETs 1 and 2, this is the entire data record. For PET 3, the next item in the data record is the following:

Item	Data Type
Attribute control flag	Integer. One of: • GKS$K_ACF_CURRENT (0) • GKS$K_ACF_SPECIFIED (1)

For PET 3, the remainder of the data record depends on the setting of Attribute Control Flag. If Attribute Control Flag equals GKS$K_ACF_SPECIFIED (1), the remainder of the data record for PET 3 is as follows:

Item	Data Type
Marker Type ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Markersize Scale Factor ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Polymarker Color Index ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Polymarker Index	Integer. A defined polymarker bundle index.
Markertype Index	Integer. A defined markertype.
Markersize Scale Factor	Real. The markersize scale factor.
Polymarker Color Index	Integer. A defined color bundle index.

If Attribute Control Flag equals GKS$K_ACF_CURRENT (0), the current polyline attributes at STROKE initialization are used.

For PET 4, the next item in the data record is the following:

Item	Data Type
Attribute control flag	Integer. One of: • GKS$K_ACF_CURRENT (0) • GKS$K_ACF_SPECIFIED (1)

For PET 4, the remainder of the data record depends on the setting of Attribute Control Flag. If Attribute Control Flag equals GKS$K_ACF_SPECIFIED (1), the remainder of the data record is as follows.

Item	Data Type
Linetype ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Linewidth Scale Factor ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Polyline Color Index ASF	Integer. One of: • GKS$K_ASF_BUNDLED (0) • GKS$K_ASF_INDIVIDUAL (1)
Polyline Index	Integer. A defined polyline bundle index.
Linetype Index	Integer. A defined linetype.
Linewidth Scale Factor	Real. The linewidth scale factor.
Polyline Color Index	Integer. A defined color bundle index.

If Attribute Control Flag equals GKS$K_ACF_CURRENT (0), the current polyline attributes at STROKE initialization are used.

Initialize Stroke

Required

Required for OUTIN workstations and for INPUT workstations that support STROKE input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The stroke logical device number.
XFORM	Integer. The transformation used to convert the initial stroke from WC to NDC. It is not used by the handler, but must be stored for return in the Inquire Stroke Device State function.
NUM_INIT_POINTS	Integer. The number of points in the initial stroke array.
INITX_ARRAY	Array of reals. The X component of the initial stroke, in NDC.
INITY_ARRAY	Array of reals. The Y component of the initial stroke, in NDC.
PROMPT_ECHO_TYPE	Integer value of the desired prompt and echo type.
ECHO_AREA	A [1 x 4] array of real numbers defining the allowable echo array in LDC. Ordered as XMIN, XMAX, YMIN, YMAX.
DATA_REC_SIZE	Integer. The number of bytes in the data record array.
STK_DATAREC	The stroke data record passed by reference.
ATTRIB_ARRAY	The attribute array, defined in Section 4.1. It is supplied to give the handler the information necessary to perform the various PETs.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_60	Polyline index is invalid.
GKS$_ERROR_63	Linetype is equal to zero.
GKS$_ERROR_65	Linewidth scale factor is less than zero.
GKS$_ERROR_66	Polymarker index is invalid.
GKS$_ERROR_67	A representation for the specified polymarker index has not been defined on this workstation.
GKS$_ERROR_92	Color index is less than zero.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Specified input device is not in REQUEST mode.
GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation.
GKS$_ERROR_145	Echo area is outside display space.
GKS$_ERROR_146	Contents of input data record are invalid.
GKS$_ERROR_152	Initial value is invalid.
GKS$_ERROR_153	Number of points in the initial stroke is greater than the buffer size.

Initialize Valuator

This function initializes the logical input device for input of type VALUATOR. It should store the input parameters in the WSL entries of the specified VALUATOR logical input device.

The GKS standard specifies the following prompt and echo types for VALUATOR input:

PET	Definition
Less than 0	Prompting and echoing is VALUATOR device dependent.
1	Designate the current VALUATOR value using an implementation-defined technique.
2	Display a graphical representation of the current VALUATOR value, such as a dial or pointer, within the echo area.
3	Display a digital representation of the current VALUATOR value within the echo area.
4 or greater	Reserved for registration or future standardization.

The data record for the user-defined PETs is undefined. For all other PETs, the data record must contain at least the following two values:

Item	Data Type
Low Value	Real. The low value of the valuator range.
High Value	Real. The high value of the valuator range.

Required

Required for OUTIN workstations and for INPUT workstations that support VALUATOR input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The valuator device number.
INIT_VALUE	Real. The initial value of the valuator.
PROMPT_ECHO_TYPE	Integer. The desired prompt and echo type.
ECHO_AREA	A [1 x 4] array of real numbers defining the allowable echo array in LDC. Array is ordered as XMIN, XMAX, YMIN, YMAX.
DATA_REC_SIZE	Integer. The number of bytes in the data record array.
VAL_DATAREC	The locator data record, passed by reference. The data record is passed straight through from the kernel to the handler.
ATTRIB_ARRAY	Array. The attribute array defined in Section 4.1. It is supplied to give the handler the information necessary to perform the various PETs.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Specified input device is not in REQUEST mode.
GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation.
GKS$_ERROR_145	Echo area is outside display space.
GKS$_ERROR_146	Contents of input data record are invalid.
GKS$_ERROR_152	Initial value is invalid.

Initialize Choice

This function initializes the logical input device for input of type CHOICE. It should store the input parameters in the WSL entries for the specified CHOICE logical input device. If the number of choices in the data record exceeds the maximum number of choice alternatives listed in the WDT, the handler must return an error.

The GKS standard specifies the following prompt and echo type definitions for CHOICE input:

PET	Definition
Less than 0	Prompting and echoing is CHOICE device dependent.
1	Designate the current CHOICE number using an implementation-defined technique.
2	Use a built-in prompt supplied by the device. The choice data record should contain a prompt array structure that specifies which device-specific prompt capability to use. For example, the array may specify which subset of a set of choice buttons to light up.
3	Let the user choose by selecting a choice string using an implementation-defined technique. In this case the choice data record should contain the number of choice strings, and the actual choice strings. The strings are displayed in the echo area. The return value is the number of the string selected.
4	Let the user make a choice by selecting a choice string using an alphanumeric keyboard. The function should display the choice strings in the echo area, then prompt the user to type one of the strings. The input the user types should be echoed in the echo area. The function should return the number of the first string in the choice data record that matches the user's input.

PET	Definition
5	Let the user make a choice by selecting a primitive (or a set of primitives) from the segment named in the choice data record. The function should display the segment by mapping the unit square (0,1) by (0,1) in NDC space to the echo area. The PICK identifiers in the segment should be mapped to the choice numbers in a device-dependent manner, so that picking the primitives associated with a pick identifier makes the function return a choice number. After the user chooses, there should be no logical connection between the segment and the choice device.
6 or greater	Reserved for registration or future standardization.

The data record depends on the PET. Data records for user-defined PETs and PET 1 are undefined. The data records for the remaining PETs are as follows:

For PET 2:

Item	Data Type
Number of choice alternatives	Integer. The number of choices.
Array of prompts	Array of integer flags. Each flag corresponds to one choice alternative. GKS$K_TRUE (1) means the choice should be active (in the case of lit buttons, the corresponding button should be lit). GKS$K_FALSE (0) means the choice should not be active.

For PETs 3 and 4:

Item	Data Type
Number of choice strings	Integer. The number of choices.
Array of choice strings	The choice strings.

For PET 5:

Item	Data Type
Segment name	The segment the user may pick from.
Number of choice alternatives	Integer. The number of choices available.
Array of pick identifiers	Array of integer. The pick identifiers within the segment.

Required

Required for OUTIN workstations and for INPUT workstations that support CHOICE input.

Input Parameters

WSL	Address. The address of the handler's local data area.
DEVNUM	Integer. The choice device number.
INIT_STATUS	Integer. The initial status. One of: • GKS$K_STATUS_NONE (0) • GKS$K_STATUS_OK (1) • GKS$K_STATUS_NOCHOICE (2)
INIT_CHOICE	Integer. The initial choice number.
PROMPT_ECHO_TYPE	Integer. The desired prompt and echo type.
ECHO_AREA	A [1 x 4] array of real numbers defining the allowable echo array in LDC. Ordered as XMIN, XMAX, YMIN, YMAX.
DATA_REC_SIZE	Integer. Number of bytes in the data record array.
CHOICE_DATAREC	Address. The choice data record, passed by reference. The data record is passed straight through from the kernel to the handler.

Initialize Choice

ATTRIB_ARRAY	The attribute array defined in Section 4.1. It is supplied to give the handler the information necessary to perform the various PETs.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Specified input device is not in REQUEST mode.
GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation.
GKS$_ERROR_145	Echo area is outside display space.
GKS$_ERROR_146	Contents of input data record are invalid.
GKS$_ERROR_152	Initial value is invalid.
GKS$_ERROR_120	Specified segment name is invalid.
GKS$_ERROR_122	Specified segment does not exist.
GKS$_ERROR_123	Specified segment does not exist on specified workstation.

Initialize String

This function initializes the workstation for input of type STRING. It should store the input parameters in the WSL for the string device.

The GKS standard specifies the following prompt and echo type definitions for STRING input:

PET	Definition
Less than 0	Prompting and echoing is STRING device dependent.
1	Display the current STRING value in the echo area.
2 or greater	Reserved for registration or future standardization.

The data record for user-defined PETs is undefined. For PET 1, the data record should contain at least the following:

Item	Data Type
Input Buffer Size	Integer. The size in bytes of the input buffer.
Initial Cursor Position	Integer. The initial cursor position.

Required

Required for OUTIN workstations and for INPUT workstations that support STRING input.

Initialize String

Input Parameters

WSL	Address. The address of the handler's local data area.
DEVNUM	Integer. The string device number.
INIT_STRING	String. The initial string, passed by descriptor.
PROMPT_ECHO_TYPE	Integer. The desired prompt and echo type.
ECHO_AREA	A [1 x 4] array of real numbers defining the allowable echo area in LDC. Ordered as XMIN, XMAX, YMIN, YMAX.
DATA_REC_SIZE	Integer. Number of bytes in the data record array.
STRING_DATAREC	The string data record passed by reference. The data record is passed straight through from the kernel to the handler.
ATTRIB_ARRAY	The attribute array defined in Section 4.1. It is supplied to give the handler the information necessary to perform the various PETs.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.
GKS$_ERROR_141	Specified input device is not in REQUEST mode.
GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation.
GKS$_ERROR_145	Echo area is outside display space.
GKS$_ERROR_146	Contents of input data record are invalid.
GKS$_ERROR_152	Initial value is invalid.

Initialize Pick

This function initializes the device for PICK mode input. It should store the input parameters in the WSL for the device.

The GKS standard specifies the following prompt and echo type definitions for PICK input:

PET	Definition
Less than 0	Prompting and echoing is pick device dependent.
1	Use an implementation-defined technique that at least highlights the picked primitive for a short period of time.
2	Echo the contiguous group of primitives within the segment with the same pick identifier as the picked primitive, or all primitives of the segment with the same pick identifier as the picked primitive.
3	Echo the whole segment containing the picked primitive.
4 or greater	Reserved for registration or future standardization.

There are no predefined data records for any PICK input PET. However, DEC GKS suggests that the first element of the data record be the aperture, or the distance around the point the user picks that may contain the segment. This should be expressed in NDC.

See Appendix C, Pick Simulation Functions, for simulation functions you can use to simplify this routine.

Required

Required for OUTIN workstations and for INPUT workstations that support PICK input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The logical input device number.
INIT_STATUS	Integer. The initial pick status. Either: • GKS$K_STATUS_OK (1) • GKS$K_STATUS_NOPICK (2)
INIT_SEGMENT	Integer. The initial segment name.
INITIAL_PICKID	Integer. The initial pick identifier.
PROMPT_ECHO_TYPE	Integer. The desired prompt and echo type.
ECHO_AREA	A [1 x 4] array of real numbers defining the allowable echo array in LDC. The array is ordered as XMIN, XMAX, YMIN, YMAX.
DATA_REC_SIZE	Integer. The number of bytes in the data record.
PICK_DATAREC	The pick data record, passed by reference. The data record is passed straight through from the kernel to the handler.
ATTRIB_ARRAY	The entire attribute array as described in Section 4.1. It is supplied to give the handler the information necessary to perform the various PETs.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Input device is not in REQUEST mode.
GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation.
GKS$_ERROR_145	Echo area is outside display space.
GKS$_ERROR_146	Contents of input data record are invalid.
GKS$_ERROR_152	Initial value is invalid.

Set Locator Mode

This function sets the specified LOCATOR device to the OPMODE specified in the input parameters, and sets echoing on or off depending on the value of ECHO_SWITCH. It should also store the OPMODE and ECHO_SWITCH in the device's WSL.

In addition, you may let your Set Locator Mode function perform other operations depending on the input mode. For example, in SAMPLE mode, your function might start an AST routine that performs prompting and echoing on the logical input device, and stores the current locator measure. In this case your Sample Locator function would only need to return the measure.

For event mode, your function might start an AST that will prompt for input, maintain the current locator measure, then call the built-in function GKS$STORE_EVENTS with the current measure when it receives the event trigger.

For simplicity, your Set Locator Mode function can start a single AST for either SAMPLE or EVENT mode. In this case the AST will ignore triggers while operating in SAMPLE mode.

Required

Required for OUTIN workstations and for INPUT workstations that support LOCATOR input.

Set Locator Mode

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The locator device number.
OPMODE	Integer. The input mode. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer value that determines whether or not to echo the input. One of: • GKS$K_NOECHO (0) • GKS$K_ECHO (1)

Output Parameters

OLD_MODE	Integer. The input mode in effect before you changed it.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Set Stroke Mode

This function sets the specified STROKE device to the OPMODE specified in the input parameters, and sets echoing on or off depending on the value of ECHO_SWITCH. It should also store the OPMODE and ECHO_SWITCH in the STROKE device's WSL.

In addition, you may let your Set Stroke Mode function perform other operations depending on the input mode. For example, in SAMPLE mode, your function might start an AST routine that performs prompting and echoing on the logical input device, and stores the current stroke measure. In this case your Sample Stroke function would only need to return the measure.

For event mode, your function might start an AST that will prompt for input, maintain the current stroke measure, then call the built-in function GKS$STORE_EVENTS with the current measure when it receives the event trigger. For both Sample and Event modes, your AST should delete the echo of the stroke points it returns. For example, if there are eight points in the stroke buffer and the GKS kernel calls Sample Stroke and requests five points, the first five points should be returned, and the echo should only show the remaining three points.

For simplicity, your Set Stroke Mode function can start a single AST for either SAMPLE or EVENT mode. In this case the AST will ignore triggers while operating in SAMPLE mode.

Required

Required for OUTIN workstations and for INPUT workstations that support STROKE input.

Set Stroke Mode

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The stroke device number.
OPMODE	Integer. The input mode. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer value that determines whether or not to echo the input. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)

Output Parameters

OLD_MODE	Integer. The input mode in effect before you changed it.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Set Valuator Mode

This function sets the specified VALUATOR device to the OPMODE specified in the input parameters, and sets echoing on or off depending on the value of ECHO_SWITCH. It should also store the OPMODE and ECHO_SWITCH in the VALUATOR device's WSL.

In addition, you may let your Set Valuator Mode function perform other operations depending on the input mode. For example, in SAMPLE mode, your function might start an AST routine that performs prompting and echoing on the logical input device, and stores the current valuator measure. In this case your Sample Valuator function would only need to return the measure.

For event mode, your function might start an AST that will prompt for input, maintain the current valuator measure, then call the built-in function GKS$STORE_EVENTS with the current measure when it receives the event trigger.

For simplicity, your Set Valuator Mode function can start a single AST for either SAMPLE or EVENT mode. In this case the AST will ignore triggers while operating in SAMPLE mode.

Required

Required for OUTIN workstations and for INPUT workstations that support VALUATOR input.

Set Valuator Mode

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The valuator device number.
OPMODE	Integer. The input mode. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer flag that determines whether or not to echo the input. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)

Output Parameters

OLD_MODE	Integer. The input mode in effect before you changed it.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Set Choice Mode

This function sets the specified CHOICE device to the OPMODE specified in the input parameters, and sets echoing on or off depending on the value of ECHO_SWITCH. It should also store the OPMODE and ECHO_SWITCH in the CHOICE device's WSL.

In addition, you may let your Set Choice Mode function perform other operations depending on the input mode. For example, in SAMPLE mode, your function might start an AST routine that performs prompting and echoing on the logical input device, and stores the current choice measure. In this case your Sample Choice function would only need to return the measure.

For event mode, your function might start an AST that will prompt for input, maintain the current choice measure, then call the built-in function GKS$STORE_EVENTS with the current measure when it receives the event trigger.

For simplicity, your Set Choice Mode function can start a single AST for either SAMPLE or EVENT mode. In this case the AST will ignore triggers while operating in SAMPLE mode.

Required

Required for OUTIN workstations and for INPUT workstations that support CHOICE input.

Set Choice Mode

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The choice device number.
OPMODE	Integer. The input mode. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer flag that determines whether or not to echo the input. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)

Output Parameters

OLD_MODE	Integer. The input mode in effect before you changed it.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Set String Mode

This function sets the specified STRING device to the OPMODE specified in the input parameters, and sets echoing on or off depending on the value of ECHO_SWITCH. It should also store the OPMODE and ECHO_SWITCH in the STRING device's WSL.

In addition, you may let your Set String Mode function perform other operations depending on the input mode. For example, in SAMPLE mode, your function might start an AST routine that performs prompting and echoing on the logical input device, and stores the current string measure. In this case your Sample String function would only need to return the measure.

For event mode, your function might start an AST that will prompt for input, maintain the current string measure, then call the built-in function GKS$STORE_EVENTS with the current measure when it receives the event trigger. For both Sample and Event modes, your AST should delete the echo of the characters it returns. For example, if there are eight characters in the string buffer and the GKS kernel calls Sample String and requests five characters, the first five points should be returned, and the echo should only show the remaining three characters.

For simplicity, your Set String Mode function can start a single AST for either SAMPLE or EVENT mode. In this case the AST will ignore triggers while operating in SAMPLE mode.

Required

Required for OUTIN workstations and for INPUT workstations that support STRING input.

Set String Mode

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The string device number.
OPMODE	Integer. The input mode. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer flag that determines whether or not to echo the input. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)

Output Parameters

OLD_MODE	Integer. The input mode in effect before you changed it.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Set Pick Mode

This function sets the specified PICK device to the OPMODE specified in the input parameters, and sets echoing on or off depending on the value of ECHO_SWITCH. It should also store the OPMODE and ECHO_SWITCH in the device's WSL.

In addition, you may let your Set Pick Mode function perform other operations depending on the input mode. For example, in SAMPLE mode, your function might start an AST routine that performs prompting and echoing on the logical input device, and stores the current pick measure. In this case your Sample Pick function would only need to return the measure.

For event mode, your function might start an AST that will prompt for input, maintain the current pick measure, then call the built-in function GKS$STORE_EVENTS with the current measure when it receives the event trigger.

Required

Required for OUTIN workstations and for INPUT workstations that support PICK input.

Set Pick Mode

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The pick device number.
OPMODE	Integer. The input mode. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer flag that determines whether or not to echo the input. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)

Output Parameters

OLD_MODE	Integer. The input mode in effect before you changed it.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Request Locator

This function gets LOCATOR input. It performs the following operations:

- If echoing is enabled, echoes the initial position and prompts for input using the current PET.
- Upon receipt of input, updates the prompt and echo to reflect new position, and stores the new location in the buffer area.
- Upon receipt of the trigger code, converts the input value to NDC and returns it.
- If the user performs a BREAK operation, the function should return the status GKS$K_STATUS_NONE (0). Otherwise, it should return GKS$K_STATUS_OK (1).

Required

Required for OUTIN workstations and for INPUT workstations that support LOCATOR input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The locator device number.

Output Parameters

LOC_X	Real. The X component of the locator point, in NDC.
LOC_Y	Real. The Y component of the locator point, in NDC.
STATUS	Integer. The input status. Either: • GKS$K_STATUS_NONE (0) • GKS$K_STATUS_OK (1)

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Specified input device is not in request mode.

Request Stroke

This function gets STROKE input. It performs the following operations:

- If echoing is enabled, echoes the initial stroke and prompts for input using the current PET.
- Upon receipt of input, updates prompt and echo to reflect new position, and stores the new coordinates in buffer area.
- Upon receipt of the trigger code, converts the stroke points to NDC, and returns the points.
- If the user performs a BREAK operation, the function should return the status GKS$K_STATUS_NONE (0). Otherwise, it should return GKS$K_STATUS_OK (1).

If the operator enters more points than the stroke input buffer size allows, the additional points are lost, and the handler should notify the user in a device-dependent manner.

Required

Required for OUTIN workstations and for INPUT workstations that support STROKE input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The stroke device number.

Modified Parameters

NUMB_PTS — Integer. The number of points stored in the two stroke arrays. On input, the size of the smallest array; on output, the number of elements written in the array.

Output Parameters

STROKE_X — Array of reals. The X components of the stroke input, in NDC.

STROKE_Y — Array of reals. The Y components of the stroke input, in NDC.

STATUS — Integer. The input status. Either:

- GKS$K_STATUS_OK (1)
- GKS$K_STATUS_NONE (0)

TOTAL_PTS — Integer. The total number of points entered in the stroke. If this is larger than NUM_PTS, the user entered more points than the arrays STROKE_X and STROKE_Y could hold.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Specified input device is not in request mode.

Request Valuator

This function gets VALUATOR input. It performs the following operations:

- If echoing is enabled, echoes the initial value and prompts for input according to the current PET.
- Upon receipt of input, updates prompt and echo to reflect new position, and updates the value in the buffer area.
- Upon receipt of the trigger code, returns the value.
- If the user performs a BREAK operation, the function should return the status GKS$K_STATUS_NONE (0). Otherwise, it should return GKS$K_STATUS_OK (1).

Required

Required for OUTIN workstations and for INPUT workstations that support VALUATOR input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer value of the valuator device number.

Output Parameters

VALUE	Real. The current value.
STATUS	Integer. The input status. Either: • GKS$K_STATUS_OK (1) • GKS$K_STATUS_NONE (0)

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Specified input device is not in request mode.

Request Choice

This function gets CHOICE input. It performs the following operations:

- If echoing is enabled and the initial status is GKS$K_STATUS_OK (1), echoes the initial choice.
- Prompts for input according to the current PET.
- Upon receipt of input, updates prompt and echo to reflect new choice, and stores it in the buffer area.
- Upon receipt of the trigger, returns the value stored in buffer area.
- If the initial status was GKS$K_STATUS_NOCHOICE (2), and no choice was made, returns the status GKS$K_STATUS_NOCHOICE (2). If the user performs a BREAK operation, the function should return the status GKS$K_STATUS_NONE (0). Otherwise, it should return GKS$K_STATUS_OK (1).

Required

Required for OUTIN workstations and for INPUT workstations that support CHOICE input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The choice device number.

Output Parameters

CHOICE	Integer. The selected choice number.
STATUS	Integer. The input status. One of:

- GKS$K_INPUT_MODE_REQUEST (0)
- GKS$K_INPUT_MODE_SAMPLE (1)
- GKS$K_INPUT_MODE_EVENT (2)

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Specified input device is not in request mode.

Request String

This function gets STRING input. It performs the following operations:

- If echoing is enabled, echoes the initial string and prompts for input according to the current PET.
- Upon receipt of input, updates prompt and echo to reflect new string, and updates the buffer area.
- Upon receipt of the trigger code, returns the value stored in buffer area by copying it to the STRING buffer area.
- If the user performs a BREAK operation, the function should return the STATUS value GKS$K_STATUS_NONE (0). Otherwise, it should return GKS$K_STATUS_OK (1).

Required

Required for OUTIN workstations and for INPUT workstations that support STRING input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer value of the string device number.

Modified Parameters

STRING_SIZE	Integer. On input, this is the size of the output string buffer in bytes; on output, this is the size of the actual string returned.

Request String

Output Parameters

STRING	Address. The string buffer, passed by reference.
STATUS	Integer. The input status. Either: • GKS$K_STATUS_NONE (0) • GKS$K_STATUS_OK (1)
TOTAL_CHAR	Integer. The total number of characters in the string. If this is greater than STRING_SIZE, the user entered more characters than the buffer STRING could hold.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_141	Specified input device is not in request mode.

Request Pick

This function gets PICK input. If your workstation supports segments, and it can perform PICK input, It performs the following operations:

- If echoing is enabled, echoes the initial pick and prompts for input according to the current PET.
- Upon receipt of input, updates prompt and echo to reflect new pick, and updates the buffer area.
- Upon receipt of the trigger code, returns the segment name and pick id.
- If the user performs a BREAK operation, the function should return the STATUS value GKS$K_STATUS_NONE (0). Otherwise, it should return GKS$K_STATUS_OK (1).

If the user picks overlapping segments, your function should return the highest-priority segment. If the overlapping segments all share the same priority, then the pick result is implementation dependent.

If the kernel is simulating segments, or your workstation cannot perform pick input, this function should get the location of an input point, similar to the REQUEST LOCATOR function. Then it should call the simulation function GKS$FIND_SEGMENT to get the segment name and pick id. GKS$FIND_SEGMENT also returns the segment extent and the pick extent, and your function can use these values for highlighting the picked segment.

See Appendix C, Pick Simulation Functions, for information about simulation functions that make writing this function easier.

Required

Required for OUTIN workstations and for INPUT workstations that support PICK input.

Request Pick

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The pick logical device number.

Output Parameters

STATUS	Integer. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
SEGMENT	Integer. The name of the picked segment.
PICKID	Integer. The pick id of the picked segment.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.
GKS$_ERROR_141	Specified input device is not in request mode.

Sample Locator

This function gets LOCATOR input. It returns the current locator position, in NDC.

Required

Required for OUTIN workstations and for INPUT workstations that support LOCATOR input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The stroke device number.

Output Parameters

LOC_X	Real. The X component of the locator position, in NDC.
LOC_Y	Real. The Y component of the locator position, in NDC.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_142	Specified input device is not in sample mode.

Sample Stroke

This function gets STROKE input. It returns points in NDC from the stroke buffer, then clears the points it returns from the stroke buffer. If you implement this function using the AST described in the Set Stroke Mode description, this function only needs to return the current values in the stroke buffer.

Your function should only remove from the stroke buffer a number of points equal to or less than the value of the parameter NUM_POINTS. This keeps the handler from losing input by clearing points from the buffer without passing them to the GKS kernel.

Also, your device handler should erase the echo of points as it passes them to the GKS kernel. For example, if there are eight points in the stroke buffer and the kernel calls Sample Stroke and requests five points, the first five points should be returned, and the echo should only show the remaining three points.

Required

Required for OUTIN workstations and for INPUT workstations that support STROKE input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The stroke device number.

Modified Parameters

NUM_POINTS	Integer. On input the size of the arrays STROKE_X and STROKE_Y. Your function should write no more than this many points to the output buffers. On output the number of elements written in the array.

Output Parameters

STROKE_X	Array of reals. The X components of the stroke input, in NDC.
STROKE_Y	Array of reals. The Y components of the stroke input, in NDC.
TOTAL_PTS	Integer. The total number of points entered in the stroke. If this is larger than NUM_PTS, the user entered more points than the arrays STROKE_X and STROKE_Y could hold.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_142	Specified input device is not in sample mode.

Sample Choice

This function gets CHOICE input. It returns the current choice. If you implement this function using the AST described in the Set Choice Mode description, this function only needs to return the current values in the choice buffer.

Required

Required for OUTIN workstations and for INPUT workstations that support CHOICE input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The choice device number.

Output Parameters

STATUS	Integer. The choice status, either GKS$K_STATUS_OK (1) or GKS$K_STATUS_NOCHOICE (0).
CHOICE_NUM	Integer. The choice number. This is valid only if STATUS is GKS$K_STATUS_OK.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_142	Specified input device is not in sample mode.

Sample Valuator

This function returns the current VALUATOR value. If you implement this function using the AST described in the Set Valuator Mode description, this function only needs to return the current values in the valuator buffer.

Required

Required for OUTIN workstations and for INPUT workstations that support VALUATOR input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The valuator device number.

Output Parameters

VALUE	Real. The current value.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_142	Specified input device is not in sample mode.

Sample String

This function returns the current string in the STRING device buffer, then clears the string that it returns from the device buffer. If you implement this function using the AST described in the Set String Mode description, this function only needs to return the current values in the string buffer.

Your function should only remove from the string buffer a number of characters equal to or less than the value of the parameter STRING_SIZE. This keeps the handler from losing input by clearing characters from the buffer without passing them to the GKS kernel.

Also, your device handler should erase the echo of characters as it passes them to the GKS kernel. For example, if there are eight characters in the stroke buffer and the GKS kernel calls Sample String and requests five characters, the first five characters should be returned, and the echo should only show the remaining three characters.

Required

Required for OUTIN workstations and for INPUT workstations that support STRING input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The stroke device number.

Modified Parameters

STRING_SIZE	Integer. On input, the size of the buffer STRING. Your function should write no more than this many characters to the output buffer. On output, the number of characters written to the buffer.

Sample String

Output Parameters

STRING	The string.
TOTAL_CHAR	Integer. The total number of characters the user entered.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_142	Specified input device is not in sample mode.

Sample Pick

This function gets PICK input. It returns the current pick value. If the kernel is simulating segments, or your workstation cannot perform pick input, this function should get the location of an input point, similar to the SAMPLE LOCATOR function. Then it should call the simulation function GKS$FIND_SEGMENT to get the segment name and pick id. GKS$FIND_SEGMENT also returns the segment extent and the pick extent, and your function can use these values for highlighting the picked segment.

See Appendix C, Pick Simulation Functions, for information about simulation functions that make writing this function easier.

Required

Required for OUTIN workstations and for INPUT workstations that support PICK input. If you implement this function using the AST described in the Set Pick Mode description, this function only needs to return the current values in the pick buffer.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The pick device number.

Output Parameters

STATUS	Integer. One of: • GKS$K_STATUS_OK (1) • GKS$K_STATUS_NOCHOICE (0)
SEGMENT_NAME	Integer. The name of the picked segment. This is valid only if STATUS is GKS$K_STATUS_OK.
PICK_ID	Integer. The pick id of the picked segment. This is valid only if STATUS is GKS$K_STATUS_OK.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.
GKS$_ERROR_140	Specified input device is not in sample mode.

Chapter 6

Workstation Handler Inquiry Functions

This chapter describes the workstation handler inquiry functions. You must provide a function for each inquiry.

All inquiries return the information listed as output parameters. If the inquiry completes successfully, it should return the status GKS$_SUCCESS. If the inquiry fails, it should return the error code that best describes the problem. In addition, if the inquiry fails, the data returned in the output parameters is implementation dependent, unless specifically noted in the function description.

Most of the errors possible in an inquiry are detected by the kernel. You do not have to test for any abnormal condition for which there is no status code specified in the function description.

Your handler only needs to respond to workstation-specific inquiries. This means that it must return information stored in your workstation's data structures. Most of these items are included in the recommended WDT and WSL structures, so parameter names in the function descriptions correspond with names in these structures whenever possible. The overview of each function tells which data structure usually contains the information that inquiry returns.

Inquiries for data from the WDT must be able to return information without the workstation being open. If the workstation is not open, the WDT is not present in the local data area. Therefore, your inquiry functions must either contain the data within their own code or be able to access the WDT in some way.

Note that because you design your own means of storing data for your handler, you must also develop your own method of returning the data. If you keep data in coherent structures, most of your inquiries need only index to the correct place in a table.

Because inquiries do not alter graphic data or workstation information in any way, you have a lot of choice in how you implement them. The sole requirement is that your inquiry functions be able to identify the data the kernel needs from the input parameters, and return the correct response in the output parameters.

Inquire List of Polyline Indexes

This inquiry returns the number of polyline bundles defined, and a list of defined indexes. This information is normally maintained in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.

Modified Parameters

NUM_PLINE_ENTRIES	Integer. On input, this is maximum number of indexes that the function can write to LIST_PLINE_IND. On output, this is the number of entries the handler has written to LIST_PLINE_IND.

Output Parameters

LIST_PLINE_IND	Integer array. The list of indexes.
TOTAL_NUM_IND	Integer. The total number of defined polyline bundles. If this is greater than NUM_PLINE_ENTRIES, not all the indexes have been written to LIST_PLINE_IND.

Inquire List of Polyline Indexes

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Polyline Representation

This inquiry returns information about the polyline bundle associated with the polyline index you specify as an input parameter. This information is normally maintained in the WSL. If the specified index is not available and SET_REALIZED is GKS$K_REALIZED (1), the function should return information for index 1.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL
The address of the handler's local data area.

PLINE_INDEX
Integer. The polyline index.

SET_REALIZED

- GKS$K_SET (0)
- GKS$K_REALIZED (1)

Integer. GKS$K_SET (0) means the function should return the value exactly as it was specified in the last call to the SET POLYLINE REPRESENTATION function. GKS$K_REALIZED (1) means to return the actual value the workstation uses. For example, the set linewidth scale factor may be 2.7, but if your device can only use whole-number scale factors, the realized value may have been rounded up to 3.0.

Inquire Polyline Representation

Output Parameters

LINE_TYPE	Integer. The linetype in the bundle.
LINEWIDTH_SCALE_ FACTOR	Real. The linewidth scale factor in the bundle.
COLOR_INDEX	Integer. Index into the color array.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_60	Polyline index is invalid.
GKS$_ERROR_61	A representation for the specified polyline index has not been defined on this workstation.

Inquire List of Polymarker Indexes

This inquiry returns the number of polymarker bundles defined for the workstation, and a list of the defined polymarker indexes. This information is normally maintained in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.

Modified Parameters

NUM_PMARK_ ENTRIES	Integer. On input, this is maximum number of indexes that the function can write to LIST_PMARK_IND. On output, this is the number of entries the handler has written to LIST_PMARK_IND.

Output Parameters

LIST_PMARK_IND	Integer array. The list of defined indexes.
TOTAL_NUM_IND	Integer. The number of polymarkers defined. If this is larger than NUM_PMARK_ENTRIES, LIST_PMARK_IND does not contain all the defined indexes.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Polymarker Representation

This inquiry returns information about the polymarker bundle associated with the polymarker index specified in the input parameters. This information is usually maintained in the WSL. If the specified index is not available and SET_REALIZED is GKS$K_REALIZED (1), the function should return information for index 1.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL

The address of the handler's local data area.

PMARK_INDEX

Integer. The polymarker bundle that you want information about.

SET_REALIZED

- GKS$K_SET (0)
- GKS$K_REALIZED (1)

Integer. GKS$K_SET (0) means the function should return the value exactly as it was specified in the last call to the SET POLYMARKER REPRESENTATION function. GKSK_GKSK_REALIZED (1) means to return the actual value the workstation uses. For example, the set linewidth scale factor may be 2.7, but if your device can only use whole-number scale factors, the realized value may have been rounded up to 3.0.

Output Parameters

MARKER_TYPE	Integer. The polymarker used in the bundle.
MSIZE_SCALE_ FACTOR	Real. The marker size scale factor used in the bundle.
COLOR_INDEX	Integer. The bundle's index into the color table.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_66	Polymarker index is invalid.
GKS$_ERROR_67	A representation for the specified polymarker index has not been defined on this workstation.

Inquire List of Text Indexes

This inquiry returns the number of text bundles defined and a list of the text indexes. This information is usually maintained in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.

Modified Parameters

NUM_TEXT_ENTRIES	Integer. On input, this is maximum number of indexes that the function can write to LIST_TEXT_IND. On output, this is the number of entries the handler has written to LIST_TEXT_IND.

Output Parameters

LIST_TEXT_IND	Integer array. The list of text indexes.
TOTAL_NUM_IND	Integer. The total number of defined text bundles. If this is greater than NUM_TEXT_ENTRIES, LIST_TEXT_IND does not contain all the text indexes.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Text Representation

This inquiry returns information about the text bundles associated with the index you specify in the input parameters. This information is normally maintained in the WSL. If the specified index is not available and SET_REALIZED is GKS$K_REALIZED (1), the function should return information for index 1.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
TEXT_INDEX	Integer. The index of the text bundle you want information about.
SET_REALIZED	• GKS$K_SET (0) • GKS$K_REALIZED (1) Integer. GKS$K_SET (0) means the function should return the value exactly as it was specified in the last call to the SET TEXT REPRESENTATION function. GKS$K_ GKS$K_REALIZED (1) means to return the actual value the workstation uses. For example, the set linewidth scale factor may be 2.7, but if your device can only use whole-number scale factors, the realized value may have been rounded up to 3.0.

Output Parameters

FONT	Integer. The font used in the bundle.
PREC	Integer. The precision used in the bundle.
CHAR_EXP_FACTOR	Real. The bundle's character expansion factor.
CHAR_SPACE	Real. The bundle's character spacing.
COLOR_INDEX	Integer. The bundle's index into the color table.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_72	Text index is invalid.
GKS$_ERROR_73	A representation for the specified text index has not been defined on this workstation.

Inquire Text Extent

This function calculates the text extent box and concatenation point for the string specified in the input parameters, with the specified text precision and text attributes. Note that clipping is not performed on the extent rectangle.

The GKS kernel passes the Attribute Source Flags (ASFs) to this function. Your function must check the ASFs to determine whether any nongeometric attributes are bundled. If they are, the function must get the bundled values from the bundle table. See Chapter 9, Workstation Handler Output Functions, for information about the ASFs.

For precisions GKS$K_TEXT_PRECISION_STRING (0) and GKS$K_TEXT_PRECISION_CHAR (1), the text extent parallelogram is the minimum that completely encloses the character bodies of the string. For text paths GKS$K_TEXT_PATH_UP (2) and GKS$K_TEXT_PATH_DOWN (3), the widest character in the font is enclosed. For GKS$K_TEXT_PRECISION_STROKE (2) precision, if the character width and character base vectors are perpendicular, the extent is a rectangle.

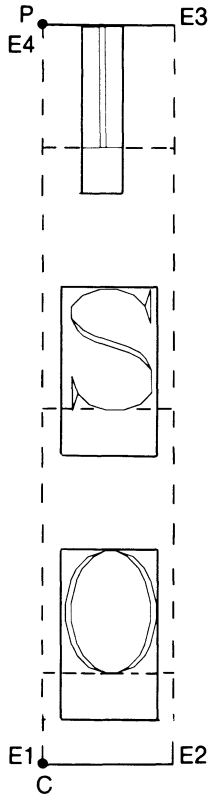
The concatenation point can be used as the origin of a subsequent text string. If text path is GKS$K_TEXT_PATH_RIGHT (0) or GKS$K_TEXT_PATH_LEFT (1), the concatenation point is displaced from the text position, in a direction determined by the horizontal component of text alignment. If the text alignment is GKS$K_HALIGN_LEFT (1), the displacement is to the right. If the text alignment is GKS$K_HALIGN_RIGHT (3), the displacement is to the left. If the text alignment is GKS$K_HALIGN_CENTER (2) or GKS$K_VALIGN_HALF (3), the displacement is zero. The size of the displacement equals the length of the sides of the text extent parallelogram parallel to the character base, plus one character space.

If text path is GKS$K_TEXT_PATH_UP (2) or GKS$K_TEXT_PATH_DOWN (3), the concatenation point is displaced from the text position in a direction determined by the vertical component of text alignment. If the component is GKS$K_VALIGN_TOP (1) or GKS$K_VALIGN_CAP (2), the displacement is down. If the component is GKS$K_VALIGN_BASE (4) or GKS$K_VALIGN_BOTTOM (5), the displacement is up. Unless the vertical component of text alignment is GKS$K_VALIGN_HALF (3), the size of the displacement equals the length of the sides of the text extent parallelogram parallel to the character up vector, plus one character space.

The effect of control characters in the character string is workstation dependent, but must be consistent with their effect in the TEXT function.

Figure 6–1, Text Extents 1, through Figure 6–5, Text Extents 5, show text extent boxes for several combinations of character spacing, character up vector, text path, and text alignment.

Figure 6–1: Text Extents 1

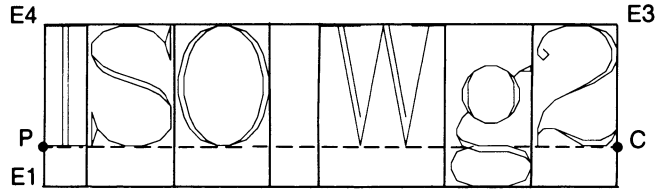


CHARACTER SPACING = 0.2
CHARACTER UP VECTOR = (0.1)
TEXT PATH = DOWN
TEXT ALIGNMENT = (LEFT.TOP)

P:	text position
C:	concatenation point
E1,E2, E3,E4:	corners of text extent rectangle, which for TEXT PATH = UP or DOWN encloses the widest character in the font

ZK-5003-86

Figure 6–2: Text Extents 2

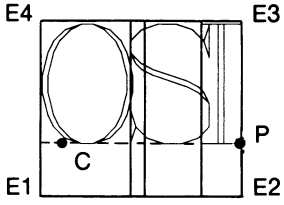


CHARACTER SPACING = 0
CHARACTER UP VECTOR = (0.1)
TEXT PATH = RIGHT
TEXT ALIGNMENT = (CENTRE.BOTTOM)

P:	text position
C:	concatenation point
E1,E2, E3,E4:	corners of text extent rectangle, which for TEXT PATH = UP or DOWN encloses the widest character in the font

ZK-5004-86

Figure 6–3: Text Extents 3

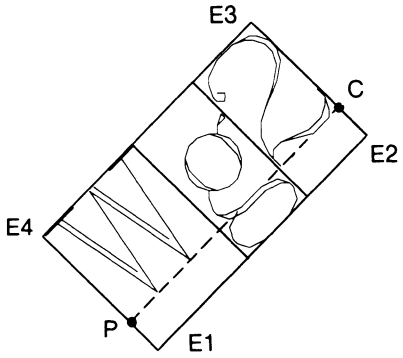


CHARACTER SPACING = -0.2
CHARACTER UP VECTOR = (0.1)
TEXT PATH = LEFT
TEXT ALIGNMENT = (RIGHT,BASE)

P:	text position
C:	concatenation point
E1,E2, E3,E4:	corners of text extent rectangle, which for TEXT PATH = UP or DOWN encloses the widest character in the font

ZK-5005-86

Figure 6–4: Text Extents 4

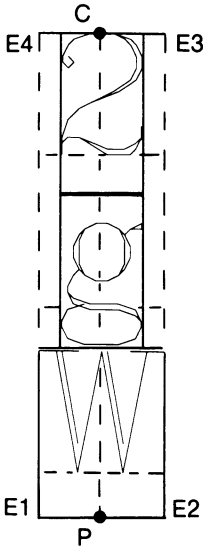


CHARACTER SPACING = 0
CHARACTER UP VECTOR = (-1.1)
TEXT PATH = RIGHT
TEXT ALIGNMENT = (NORMAL.NORMAL)

P: text position
C: concatenation point
E1,E2, E3,E4: corners of text extent rectangle, which for TEXT PATH = UP or DOWN encloses the widest character in the font

ZK-5006-86

Figure 6–5: Text Extents 5



CHARACTER SPACING = 0
CHARACTER UP VECTOR = (0,1)
TEXT PATH = UP
TEXT ALIGNMENT = (CENTER BOTTOM)

P:	text position
C:	concatenation point
E1,E2, E3,E4:	corners of text extent rectangle, which for TEXT PATH = UP or DOWN encloses the widest character in the font

ZK-5007-86

Required

Required for OUTPUT and OUTIN, workstations.

Input Parameters

WSL — Integer. The address of the handler's local data area.

TEXT_X — Real. The X value of the starting text position, in WC.

TEXT_Y — Real. The Y value of the starting text position, in WC.

STRING — Class S descriptor. The actual text character string.

ATTRIB_ARRAY — Integer array. Contains the following elements from the Active Attribute Array:

Attribute	Description
TEXT_INDEX	Integer. The index into the text bundle table. If the text index is not present in the text bundle table, the function should assume text index 1.
FONT	Integer. The text font number (either a DIGITAL-supported font, or a handler-specific font).
PREC	Integer. The text precision. One of: • GKS$K_TEXT_PRECISION_STRING (0) • GKS$K_TEXT_PRECISION_CHAR (1) • GKS$K_TEXT_PRECISION_STROKE (2)
CHAR_EXP_FACTOR	Real. The text character expansion factor.
CHAR_SPACE	Real. The text character spacing value.
COLOR_INDEX	Integer. The index into the color table.
CHARACTER_HEIGHT_X	Real. The X component of the character height vector, WC.
CHARACTER_HEIGHT_Y	Real. The Y component of the character height vector, WC.

Attribute	Description
CHARACTER_ WIDTH_X	Real. The X component of the character width vector, WC.
CHARACTER_ WIDTH_Y	Real. The Y component of the character width vector, WC.
TEXT_PATH	Integer. The direction of the text string. One of: • GKS$K_TEXT_PATH_RIGHT (0) • GKS$K_TEXT_PATH_LEFT (1) • GKS$K_TEXT_PATH_UP (2) • GKS$K_TEXT_PATH_DOWN (3)
TEXT_ ALIGNMENT_ HORZ	Integer. The horizontal text alignment. One of: • GKS$K_HALIGN_NORMAL (0) • GKS$K_HALIGN_LEFT (1) • GKS$K_HALIGN_CENTER (2) • GKS$K_HALIGN_RIGHT (3)
TEXT_ ALIGNMENT_ VERT	Integer. The vertical text alignment. One of: • GKS$K_VALIGN_NORMAL (0) • GKS$K_VALIGN_TOP (1) • GKS$K_VALIGN_CAP (2) • GKS$K_VALIGN_HALF (3) • GKS$K_VALIGN_BASE (4) • GKS$K_VALIGN_BOTTOM (5)
ASF_MASK	A 32-bit bitmask holding the attribute source flags.

Output Parameters

CONC_X	Real. The X value of the concatenation point in WC.
CONC_Y	Real. The Y value of the concatenation point in WC.
REC_X	An array of four reals describing the X coordinates of the extent rectangle in WC; point order starts with lower left and moves in a counterclockwise direction.
REC_Y	An array of four reals describing the Y coordinates of the extent rectangle in WC; point order starts with lower left and moves in a counterclockwise direction.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_101	Invalid code in string.

Inquire List of Fill Area Indexes

This inquiry returns the number of defined fill area bundles and list of fill area indexes. This information is usually maintained in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.

Modified Parameters

NUM_FILL_ENTRIES	Integer. On input, this is maximum number of indexes that the function can write to LIST_FILL_IND. On output, this is the number of entries the handler has written to LIST_FILL_IND.

Output Parameters

LIST_FILL_IND	Integer array. The list of fill area indexes.
TOTAL_NUM_IND	Integer. The number of defined fill area styles. If this number is greater than NUM_FILL_ENTRIES, then LIST_FILL_IND does not contain all the defined fill indexes.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Fill Area Representation

This inquiry returns the fill area bundle values for the index you specify as an input parameter. This information is usually maintained in the WSL. If the specified index is not available and SET_REALIZED is GKS$K_REALIZED (1), the function should return information for index 1.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL
The address of the handler's local data area.

FILL_INDEX
Integer. The fill index.

SET_REALIZED

- GKS$K_SET (0)
- GKS$K_REALIZED (1)

Integer. GKS$K_SET (0) means the function should return the value exactly as it was specified in the last call to the SET FILL AREA REPRESENTATION function. GKS$K_REALIZED (1) means to return the actual value the workstation uses. For example, if the FILL_STYLE_IND was set to be hatch style 4, and your workstation can only do hatch style 1, then the realized value would be 1.

Output Parameters

FILL_INTSTYLE
Integer. The fill area style in the bundle.

FILL_STYLE_IND
Integer. The fill style index in the bundle.

COLOR_INDEX
Integer. The bundle's index into the color table.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_80	Fill area index is invalid.
GKS$_ERROR_81	A representation for the specified fill area index has not been defined on this workstation.

Inquire List of Pattern Indexes

This inquiry returns the number of defined pattern representations and a list of the fill pattern indexes. This information is usually maintained in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.

Modified Parameters

NUM_PATT_ENTRIES	Integer. On input, this is maximum number of indexes that the function can write to LIST_PATT_IND. On output, this is the number of entries the handler has written to LIST_PATT_IND.

Output Parameters

LIST_PATT_IND	Integer array. The list of pattern indexes.
TOTAL_NUM_IND	Integer. The number of patterns defined. If this number is greater than NUM_PATT_ENTRIES, LIST_PATT_IND does not contain all the indexes.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Pattern Representation

This inquiry returns information about the pattern you specify as an input parameter. The information includes the size of the pattern, and the pattern itself represented as an array of color indexes. This information is usually maintained in the WSL. If the specified index is not available and SET_REALIZED is GKS$K_REALIZED (1), the function should return information for index 1. Pattern index 1 is present if PATTERN is supported on the workstation.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
PATT_INDEX	Integer. The pattern index.
SET_REALIZED	• GKS$K_SET (0) • GKS$K_REALIZED (1) Integer. GKS$K_SET (0) means the function should return the value exactly as it was specified in the last call to the SET PATTERN REPRESENTATION function. GKS$K_REALIZED (1) means to return the actual value the workstation uses. For example, the set number of rows may be 12, but if your device can work with no more than 10 rows, the actual NUM_ROWS would be 10.
COL_MAJOR_FLAG	Integer flag. GKS$K_TRUE (1) means PATTERN_ARRAY should be column major, and GKS$K_FALSE (0) means it should be row major.

Modified Parameters

NUM_ROWS	Integer. The number of rows in the pattern. On input this is maximum number of rows the function can write to PATTERN_ARRAY. On output, this is the number of rows the handler has written to PATTERN_ARRAY.
NUM_COLS	Integer. The number of columns in the pattern. On input this is maximum number of columns the function can write to PATTERN_ARRAY. On output this is the number of rows the handler has written to PATTERN_ARRAY.

Output Parameters

PATTERN_ARRAY	Two-dimensional array of integers describing the pattern.
PATT_DIM_X	Integer. The total number of rows in the pattern. If this value is greater than NUM_ROWS, the PATTERN_ ARRAY does not contain all the rows in the pattern.
PATT_DIM_Y	Integer. The total number of columns in the pattern. If this value is greater than NUM_COLS, the PATTERN_ ARRAY does not contain all the columns in the pattern.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_85	Specified pattern index is invalid.
GKS$_ERROR_88	A representation for the specified pattern index has not been defined on this device.
GKS$_ERROR_90	Interior style PATTERN is not supported on this workstation.

Inquire List of Color Indexes

This inquiry returns the number of defined color indexes and a list of the defined indexes. This information is usually maintained in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.

Modified Parameters

NUM_COLOR_ ENTRIES	Integer. On input, this is maximum number of indexes that the function can write to LIST_COLOR_IND. On output, this is the number of entries the handler has written to LIST_COLOR_IND.

Output Parameters

LIST_COLOR_IND	Integer array. The list of defined color indexes.
TOTAL_NUM_IND	Integer. The number of defined colors. If this is greater than NUM_COLOR_ENTRIES, LIST_COLOR_IND does not contain all the indexes.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Color Representation

This inquiry returns the color representation for the index you specify as an input parameter. The information is usually maintained in the WSL. If the specified index is not available and SET_REALIZED is GKS$K_REALIZED (1), the function should return information for index 1.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
COLOR_INDEX	Integer.
SET_REALIZED	• GKS$K_SET (0) • GKS$K_REALIZED (1) Integer. GKS$K_SET (0) means the function should return the value exactly as it was specified in the last call to the SET COLOR REPRESENTATION function. GKS$K_REALIZED (1) means to return the actual value the workstation uses. For example, the set color index may contain the RED values 0.75. If your workstation can only accept values with one digit right of the decimal, it might return the value 0.7.

Output Parameters

RED	Real. The red intensity.
GREEN	Real. The green intensity.
BLUE	Real. The blue intensity.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_93	Color index is invalid.
GKS$_ERROR_94	A representation for the specified color index has not been defined on this workstation.

Inquire Workstation Transformation

This inquiry returns information about the current and requested workstation transformation. This information includes whether or not a transformation is pending for the workstation, the current workstation window and viewport, and the requested workstation window and viewport. The information is usually maintained in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.

Output Parameters

TRANSFORM_FLAG	Integer flag. Either: • GKS$K_PENDING (1) • GKS$K_NOTPENDING (0)
REQ_WS_WINDOW	Array of 4 reals. The requested workstation window, ordered as XMIN, XMAX, YMIN, YMAX; all in NDC.
CUR_WS_WINDOW	Array of 4 reals. The current workstation window, ordered as XMIN, XMAX, YMIN, YMAX; all in NDC.
REQ_WS_VIEWPORT	Array of 4 reals. The requested workstation viewport, ordered as XMIN, XMAX, YMIN, YMAX; all in LDC.
CUR_WS_VIEWPORT	Array of 4 reals. The current workstation viewport, ordered as XMIN, XMAX, YMIN, YMAX; all in LDC.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Locator Device State

This inquiry returns information about the locator input device you specify. The information is usually maintained in the WSL.

Required

Required for OUTIN and INPUT workstations that support LOCATOR input.

Input Parameters

WSL
: The address of the handler's local data area.

DEVNUM
: Integer. The locator logical input device number.

SET_REALIZED
: - GKS$K_SET (0)
- GKS$K_REALIZED (1)

Integer. GKS$K_SET (0) means the function should return the data record exactly as it was specified in the last call to SET LOCATOR MODE. GKS$K_REALIZED (1) means to return the actual data record that the workstation uses.

Modified Parameters

DATA_REC_SIZE
: Integers. The size of the data record, in bytes. On input this is the maximum number of bytes your function can write to LOC_DATA_RECORD. On output, this is the number of bytes written into the data record.

Output Parameters

OPMODE	Integer. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer. Either: • GKS$K_NOECHO (0) • GKS$K_ECHO (1)
XFORM	Integer. The transformation used to convert points from WC to NDC, as set in INITIALIZE_LOCATOR.
INIT_LOCN_X	Real. The X component of the initial locator position, in NDC.
INIT_LOCN_Y	Real. The Y component of the initial locator position, in NDC.
PROMPT_ECHO_TYPE	Integer. The prompt and echo type.
ECHO_AREA	Array of 4 reals. The corners of the echo area in LDC, in the order XMIN, XMAX, YMIN, YMAX.
LOC_DATA_RECORD	Array. The locator data record as defined in Chapter 5. If SET_REALIZED = GKS$K_SET (0), this must be the data record which was passed to the handler at the last call to INITIALIZE_LOCATOR. For SET_REALIZED = GKS$K_REALIZED (1), this must be the full data record for the current PET, with set values replaced by realized values. If the set data record was incomplete (for example, if the application omitted some fields), the realized data record must include the defaults used in place of the unspecified values.
TOTAL_DATA_SIZE	Integer of the total size of the data record. If this is larger than DATA_REC_SIZE, LOC_DATA_RECORD was not big enough to hold the entire data record, so the entire data record was not written.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Inquire Stroke Device State

This inquiry returns information about the stroke input device you specify. This information is usually maintained in the WSL.

Required

Required for OUTIN and INPUT workstations that support STROKE input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The stroke logical input device number.
SET_REALIZED	• GKS$K_SET (0) • GKS$K_REALIZED (1) Integer. GKS$K_SET (0) means the function should return the data record exactly as it was specified in the last call to SET STROKE MODE. GKS$K_REALIZED (1) means to return the actual data record that the workstation uses.

Modified Parameters

INIT_PTS	Integer. The number of the points in the points array. On input this is the maximum number of points your function may write to the points array. On output, this is the number of points written in the points array
DATA_REC_SIZE	Integer. The size of the data record, in bytes. On input, this is the maximum number of bytes your function can write to STROKE_DATA_RECORD. On output, this is the number of bytes written into the data record.

Output Parameters

OPMODE	Integer. One of • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer. Either: • GKS$K_NOECHO (0) • GKS$K_ECHO (1)
XFORM	Integer. The transformation used to convert the points from WC to NDC. This is set in the INITIALIZE_STROKE function.
TOTAL_PTS	Integer. The total number of points in the stroke. If this is larger than INIT_PTS, the INIT_X and INIT_Y arrays were not large enough to contain all the initial points, so some of them were not written to the array.
INITX_ARRAY	Array of reals. The X component of the initial points, in NDC.
INITY_ARRAY	Array of reals. The Y component of the initial points, in NDC.
PROMPT_ECHO_TYPE	Integer. The prompt and echo type.
ECHO_AREA	Array of 4 reals. The corners of the echo area in LDC as XMIN, XMAX, YMIN, YMAX.
STROKE_DATA_RECORD	Array. The stroke data record as defined in Chapter 5. If SET_REALIZED = GKS$K_SET (0), this must be the data record which was passed to the handler at the last call to INITIALIZE_STROKE. For SET_REALIZED = GKS$K_REALIZED (1), this must be the full data record for the current PET, with set values replaced by realized values. If the set data record was incomplete (for example, if the application omitted some fields), the realized data record must include the defaults used in place of the unspecified values.

TOTAL_DATA_SIZE	Integer of the total size of the data record. If this is larger than DATA_REC_SIZE, STROKE_DATA_RECORD was not big enough to hold the entire data record, so the entire data record was not written.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Inquire Valuator Device State

This inquiry returns information about the valuator input device you specify. The information is usually maintained in the WSL.

Required

Required for INPUT and OUTIN workstations that support VALUATOR input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The valuator logical input device number.

Modified Parameters

DATA_REC_SIZE	Integer. The size of the data record, in bytes. On input this is the maximum number of bytes your function can write to VAL_DATA_RECORD. On output, this is the number of bytes written into the data record.

Output Parameters

OPMODE	Integer. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer. Either: • GKS$K_NOECHO (0) • GKS$K_ECHO (1)
INIT_VALUE	Real. The initial value of the valuator.
PROMPT_ECHO_TYPE	Integer. The prompt and echo type.
ECHO_AREA	Array of 4 reals. The corners of the echo area in LDC as XMIN, XMAX, YMIN, YMAX.
VAL_DATA_RECORD	Array. The valuator data record as defined in Chapter 5.
TOTAL_DATA_SIZE	Integer of the total size of the data record. If this is larger than DATA_REC_SIZE, VAL_DATA_RECORD was not big enough to hold the entire data record, so the entire data record was not written.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.

Inquire Choice Device State

This inquiry returns information about the choice input device you specify. The information is usually maintained in the WSL.

Required

Required for INPUT and OUTIN workstations that support CHOICE input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The choice logical input device number.

Modified Parameters

DATA_REC_SIZE	Integer. The size of the data record, in bytes. On input this is the maximum number of bytes your function can write to CHOICE_DATA_RECORD. On output, this is the number of bytes written into the data record.

Output Parameters

OPMODE	Integer. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer. Either: • GKS$K_NOECHO (0) • GKS$K_ECHO (1)
INITIAL_STATUS	Integer. The initial choice status, either: • GKS$K_STATUS_OK (1) • GKS$K_STATUS_NOCHOICE (2)
INITIAL_CHOICE	Integer. The value of the initial choice.
PROMPT_ECHO_TYPE	Integer. The prompt and echo type used.
ECHO_AREA	Array of 4 reals. The corners of the echo area in LDC as XMIN, XMAX, YMIN, YMAX.
CHOICE_DATA_RECORD	Array. The choice data record as defined in Chapter 5.
TOTAL_DATA_SIZE	Integer of the total size of the data record. If this is larger than DATA_REC_SIZE, CHOICE_DATA_RECORD was not big enough to hold the entire data record, so the entire data record was not written.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.

Inquire String Device State

This inquiry returns information about input devices of type string on the workstation. The information is usually maintained in the WSL.

Required

Required for OUTIN and INPUT workstations that support STRING input.

Input Parameters

WSL	The address of the handler's local data area.
DEVNUM	Integer. The string logical input device number.

Modified Parameters

INIT_STRING_SIZE	Integer. On input, this is the maximum number of bytes the function can write to the string buffer area. On output, this is the number of characters written into the buffer.
DATA_REC_SIZE	Integer. The size of the data record, in bytes. On input this is the maximum number of bytes your function can write to STRING_DATA_RECORD. On output, this is the number of bytes written into the data record.

Output Parameters

OPMODE	Integer. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer. One of: • GKS$K_NOECHO (0) • GKS$K_ECHO (1)
TOTAL_STRING_SIZE	Integer. The total number of bytes in the string. If TOTAL_STRING_SIZE is larger than INIT_STRING_SIZE, then not all the initial string was copied into INIT_STRING.
INIT_STRING	The string buffer, passed by reference.
PROMPT_ECHO_TYPE	Integer. The prompt and echo type.
ECHO_AREA	Array of 4 reals. The corners of the echo area in LDC. Ordered as XMIN, XMAX, YMIN, YMAX.
STRING_DATA_RECORD	Array. The string data record as defined in Chapter 5.
TOTAL_DATA_SIZE	Integer of the total size of the data record. If this is larger than DATA_REC_SIZE, STRING_DATA_RECORD was not big enough to hold the entire data record, so the entire data record was not written.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on workstation.

Inquire Pick Device State

This inquiry returns information about an input device of type pick on your workstation. This information is usually maintained in the WSL.

Required

Required for OUTIN and INPUT workstations that support PICK input.

Input Parameters

WSL
: The address of the handler's local data area.

DEVNUM
: Integer. The pick logical input device number.

SET_REALIZED
: - GKS$K_SET (0)
- GKS$K_REALIZED (1)

Integer. GKS$K_SET (0) means the function should return the data record exactly as it was specified in the last call to SET PICK MODE. GKS$K_REALIZED (1) means to return the actual data record the workstation uses.

Modified Parameters

DATA_REC_SIZE
: Integer. The size of the data record, in bytes. On input, this is the maximum number of bytes your functions can write to PICK_DATA_RECORD. In output this is the number of bytes written to PICK_DATA_RECORD.

Output Parameters

OPMODE	Integer. One of: • GKS$K_INPUT_MODE_REQUEST (0) • GKS$K_INPUT_MODE_SAMPLE (1) • GKS$K_INPUT_MODE_EVENT (2)
ECHO_SWITCH	Integer. One of: • GKS$K_NOECHO (0) • GKS$K_ECHO (1)
INIT_STATUS	Integer. One of: • GKS$K_STATUS_OK (1) • GKS$K_STATUS_NOPICK (2)
INIT_SEGMENT	Integer. The initial segment name.
INIT_PICKID	Integer. The initial pick identifier.
PROMPT_ECHO_TYPE	Integer. The current prompt and echo type for this input device.
ECHO_AREA	Array of 4 reals. The corners of the echo area in LDC. Ordered as XMIN, XMAX, YMIN, YMAX.
PICK_DATA_RECORD	Array. The pick data record as defined in Chapter 5. If SET_REALIZED = GKS$K_SET (0), this must be the data record which was passed to the handler at the last call to INITIALIZE_PICK. For SET_REALIZED = GKS$K_REALIZED (1), this must be the full data record for the current PET, with set values replaced by realized values. If the set data record was incomplete (for example, if the application omitted some fields), the realized data record must include the defaults used in place of the unspecified values.

TOTAL_DATA_SIZE	Integer of the total size of the data record. If this is larger than DATA_REC_SIZE, PICK_DATA_RECORD was not big enough to hold the entire data record, so the entire data record was not written.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Inquire Workstation Deferral and Update State

This function returns the current values of DEF_MODE,REGEN_MODE, DISPLAY_EMPTY, and NEW_FRAME. This information is usually maintained in the WSL.

Note that if the workstation does not perform segments, the value of NEW_FRAME will not be meaningful.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSL	The address of the handler's local data area.

Output Parameters

DEF_MODE	Integer. The device deferral mode. One of: • GKS$K_ASAP (0) • GKS$K_BNIG (1) • GKS$K_BNIL (2) • GKS$K_ASTI (3)
REGEN_MODE	Integer. The implicit regeneration mode. One of: • GKS$K_IRG_SUPPRESSED (0) • GKS$K_IRG_ALLOWED (1)

DISPLAY_EMPTY — Integer. A flag indicating whether the display surface is empty. Either:

- GKS$K_NOTEMPTY (0)
- GKS$K_EMPTY (1)

NEW_FRAME — Integer. A flag indicating whether the screen must be redrawn at update. Either:

- GKS$K_FALSE (0)
- GKS$K_TRUE (1)

Status Codes

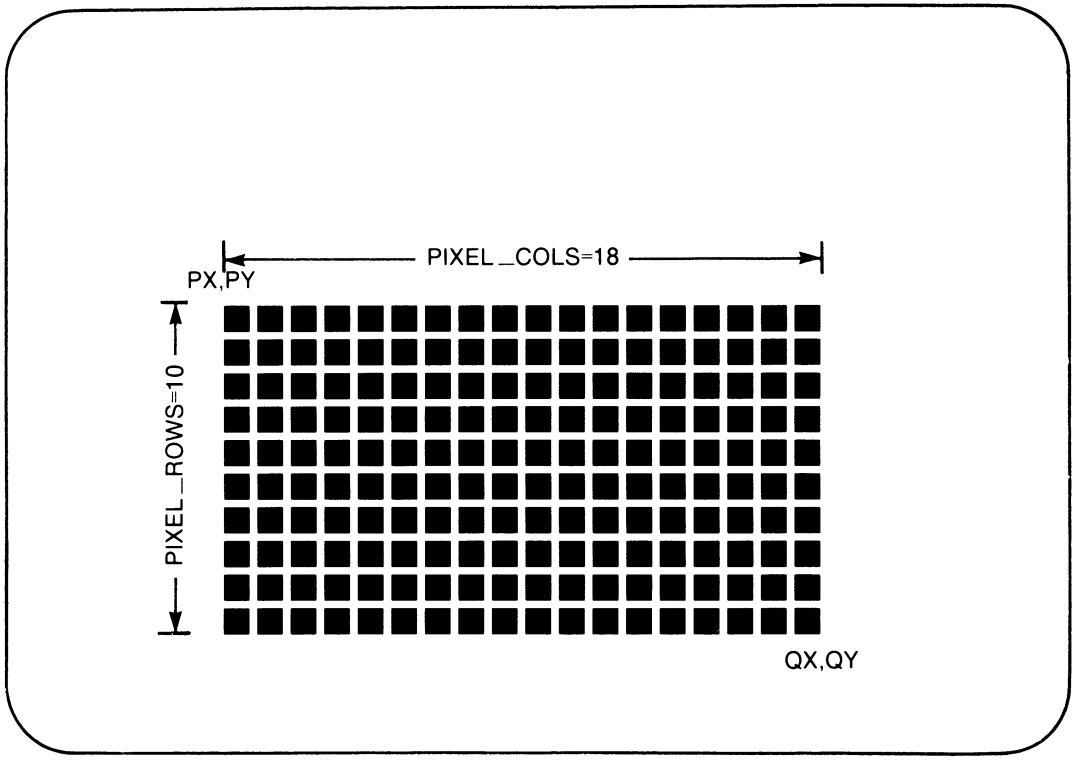
Code	Meaning
GKS$_SUCCESS	Success.

Inquire Pixel Array Dimensions

This inquiry calculates the number of columns and rows of pixels that lie within the rectangle defined by the input parameters. The input parameters P and Q define opposite corners of a rectangle in WC. There is no implicit relationship between these corners, so P may be above, below, left, or right of Q.

Figure 6–6, Pixel Array Dimensions, illustrates the pixel array dimensions.

Figure 6–6: Pixel Array Dimensions



ZK-5018-86

The handler should apply the current normalization and workstation transformation to map the rectangle onto the display surface. Then it returns the number of rows and columns of pixels within the rectangle. Note that it performs no clipping.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSL	The address of the handler's local data area.
P_X	X coordinate of the first point, in WC.
P_Y	Y coordinate of the first point, in WC.
Q_X	X coordinate of the second point, in WC.
Q_Y	Y coordinate of the second point, in WC.

Output Parameters

PIXEL_ROWS	Integer. The number of rows of pixels.
PIXEL_COLS	Integer. The number of columns of pixels.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Pixel Array

This inquiry builds a color array. The function maps the specified point onto the display surface according to the current normalization and workstation transformation. This point becomes the upper-left corner of an array of pixels with the number of rows and columns given as input parameters.

Then the function returns an array containing the color index of each pixel, oriented so that the first dimension increases with an increase along the X axis, and the second dimension increases with a decrease along the Y axis. If the function cannot determine the color index of a pixel, it returns the value -1 for that cell, and sets the INVALID_FLAG to GKS$K_TRUE (1). This might occur if the normalization and workstation transformations map part of the pixel array off the display surface.

The function should start writing the color information at the point in the array specified by the X and Y offset parameters.

If the device cannot return the pixel array it should return the error code GKS$_ERROR_40.

Required

Required for OUTIN and OUTPUT workstations.

Input Parameters

WSL	The address of the handler's local data area.
P_X	Real. The X coordinate of the reference point on the display, in WC.
P_Y	Real. The Y coordinate of the reference point on the display, in WC.
COL_MAJOR_FLAG	Integer flag. GKS$K_TRUE (1) means the array is column major and GKS$K_FALSE (0) means it is row major.

ARRAY_ROW	Integer. The index to the row of COLOR_INDEX_ARRAY where your function should begin writing.
ARRAY_COL	Integer. The index to the column of COLOR_INDEX_ARRAY where your function should begin writing.
TOTAL_ROW	Integer. The total number of rows in the color array.
TOTAL_COL	Integer. The total number of columns in the color array.
NUM_ROWS	Integer. On input, this is the number of pixel rows your function should get from the display. On output, this is the number of rows actually returned.
NUM_COLS	Integer. On input, this is the number of pixel columns your function should get from the display. On output, this is the number of rows actually returned.

Output Parameters

COLOR_INDEX_ARRAY	A two-dimensional integer array containing the colors of the pixels you selected.
INVALID_FLAG	Integer flag. If GKS$K_TRUE (1), there are invalid values in the color index array. If GKS$K_FALSE (0), no invalid values are present.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_1	Dimensions of color array are invalid.
GKS$_ERROR_40	Specified workstation has no pixel store read-back capability.

Inquire Pixel

This inquiry returns the color index of one pixel on the display. It applies the current normalization and workstation transformations to the point supplied in the input parameters and returns the color index of the pixel at that point. If the device has no pixel read-back capability it should return the error code GKS$_ERROR_40. If the device has pixel read-back capabilities but it cannot determine the color of the pixel in the input parameters (for example, if the pixel is not on the display surface), the device should return the color code -1.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSL	Integer.
P_X	X coordinate of the pixel, in WC.
P_Y	Y coordinate of the pixel, in WC.

Output Parameters

COLOR_INDEX	Integer. The pixel's index into the color table.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_40	Specified workstation has no pixel store read-back capability.

Inquire Segment Names on Workstation

This inquiry returns a list of open segments on the workstation. This information is usually kept in the WSL.

Required

Required for OUTPUT and OUTIN workstations that support segments.

Input Parameters

WSL	Integer. The address of the handler's local data area.

Modified Parameters

NUM_SEG_NAMES	Integer. On input, this is the maximum number of segment names that the function can write to SEG_LIST. On output this is the number of segment names the handler has written to SEG_LIST.

Output Parameters

SEG_LIST	Array of integers. The list of segment names on this workstation.
TOTAL_SEG_NAMES	Integer. The total number of segment names on this workstation. If this is greater than NUM_SEG_NAMES, then SEG_LIST was too small to hold all the names, and some were not written to the array.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Workstation Category

This inquiry returns the workstation category of the workstation type you identify in the input parameters. This information is normally contained in the WDT.

Required

Required for all workstations.

Input Parameters

WSTYPE	Integer. The workstation type. This WSTYPE is the full 32 bit WSTYPE, including any bitmasks.

Output Parameters

WORKSTATION_ CATEGORY	Integer. One of: • GKS$K_WSCAT_OUTPUT (0) • GKS$K_WSCAT_INPUT (1) • GKS$K_WSCAT_OUTIN (2) • GKS$K_WSCAT_MO (4) • GKS$K_WSCAT_MI (5)

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Workstation Classification

This inquiry returns the workstation classification of the workstation type you identify in the input parameter. This information is normally contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

DISPLAY_TYPE	Integer. One of: • GKS$K_WSCLASS_VECTOR (0) • GKS$K_WSCLASS_RASTER (1) • GKS$K_WSCLASS_OTHERD (2)

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Display Space Size

This inquiry returns the display surface size for the workstation type in the input parameters. It returns both the vertical and horizontal dimensions in both LDC units and raster units. This information is normally contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

DEV_COORDINATE_UNITS	Integer flag. Either: • GKS$K_METERS (0) • GKS$K_OTHER_UNITS (1)
DEV_DISPLAY_SPACE_SIZE_X	Real. The X dimension in LDC.
DEV_DISPLAY_SPACE_SIZE_Y	Real. The Y dimension in LDC.
RASTER_DISPLAY_SPACE_SIZE_X	Integer. The X dimension in raster units.
RASTER_DISPLAY_SPACE_SIZE_Y	Integer. The Y dimension in raster units.

Inquire Display Space Size

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Polyline Facilities

This inquiry returns information about the workstation type's polyline capabilities from the WDT. This includes the number of lines supported, and the maximum, minimum, and nominal linewidths.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Modified Parameters

NUM_LTYPES	Integer. On input, this is the maximum number of line types that the function can write to LIST_LINE_TYPES. On output, this is the number of entries the handler has written to LIST_LINE_TYPES.

Output Parameters

TOTAL_NUM_LTYPES	Integer. The number of linetypes your device supports. If this is larger than NUM_LTYPES, then LIST_LINE_TYPES was too small to hold all the line types, and some were not written to the array.
LIST_LINE_TYPES	Array of integers. The list of linetype bundle indexes.
NUM_LINEWIDTHS	Integer. The number of linewidths the device supports. This is 0 if the device supports a continuous range of linewidths, or the discrete number of linewidths.
NOMINAL_ LINEWIDTH	Real. The size in LDC of the linewidth the device draws when the linewidth is set to 1.
MAXIMUM_ LINEWIDTH	Real. The size in LDC of the largest linewidth the device supports.
MINIMUM_ LINEWIDTH	Real. The size in LDC of the thinnest linewidth the device supports.
NUMBER_PREDEF_ PLINE_IND	Integer. The total number of predefined linetypes.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Predefined Polyline Representation

This inquiry returns information about the specified predefined polyline representation on the workstation. This information is normally contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.
PLINE_INDEX	Integer. The polyline index.

Output Parameters

LINE_TYPE	Integer. The linetype in the bundle.
LINEWIDTH_SCALE_FACTOR	Real. The bundle's linewidth scale factor.
COLOR_INDEX	Integer. The bundle's index into the color table.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_60	Polyline index is invalid.
GKS$_ERROR_62	A representation for the specified polyline index has not been predefined on this workstation.

Inquire Polymarker Facilities

This inquiry returns information about the polymarker capabilities of the workstation type you specify, including the marker types it supports, and the maximum, minimum, and nominal marker sizes. This information is normally contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Modified Parameters

NUM_MTYPES	Integer. On input, this is the maximum number of marker types that the function can write to LIST_MARKERTYPES. On output, this is the number of entries the handler has written to LIST_MARKERTYPES.

Output Parameters

NUM_MARKERTYPES	Integer. The number of linetypes your device supports. If this is larger than NUM_MTYPES, then LIST_MARKERTYPES was too small to hold all the line types, and some were not written to the array.
LIST_MARKERTYPES	Array of integers. The marker types the device supports.
NUM_MSIZES	Integer. The number of marker sizes the device supports. This is 0 if the device supports a continuous range, or the discrete number of sizes.
NOMINAL_MSIZE	Real. The size in LDC of the marker the device draws with marker size = 1.
MINIMUM_MSIZE	Real. The size in LDC of the smallest marker the device supports.
MAXIMUM_MSIZE	Real. The size in LDC of the largest marker the device supports.
NUMBER_PREDEF_PMARK_IND	Integer. The number of predefined marker types.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Predefined Polymarker Representation

This inquiry returns information about the predefined polymarker representation you specify on the workstation type you specify in the input parameters. This information is usually contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.
PMARK_INDEX	Integer. The index into the polymarker table.

Output Parameters

MARKER_TYPE	Integer. The marker type in the bundle.
MSIZE_SCALE_ FACTOR	Real. The marker size in the bundle.
COLOR_INDEX	Integer. The index into the color table.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_66	Polymarker index is invalid.
GKS$_ERROR_68	A representation for the specified polymarker index has not been predefined on this workstation.

Inquire Text Facilities

This inquiry returns information about the text capabilities of the device you specify. This data is usually stored in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Modified Parameters

NUM_FONT_PREC	Integer. On input, this is the maximum number of font-precision pairs that the function can write to LIST_FONT and LIST_PREC. On output, this is the number of entries the handler has written to LIST_FONT and LIST_PREC.

Output Parameters

NUM_FONT_PREC_ PAIRS	Integer. The number of font-precision pairs your device supports. If this is larger than NUM_FONT_PREC, then LIST_FONTS and LIST_PREC were too small to hold all the font-precision pairs, and some were not written to the arrays.
LIST_FONT	Array of integers. The font component of the font-precision pairs the device supports.
LIST_PREC	Array of integers. The precision component of the font-precision pairs the device supports.
NUM_CHAR_HEIGHTS	Integer. The number of character heights the device supports for Font 1. This is 0 if the device supports a continuous range of heights, or the discrete number.
MINIMUM_CHAR_ HEIGHT	Real. The size in LDC of the smallest height the device can draw for Font 1.
MAXIMUM_CHAR_ HEIGHT	Real. The size in LDC of the largest height the device can draw for Font 1.
NUM_CHAR_EXP_ FACTOR	Integer. The number of character expansion factors the device supports for Font 1. This is 0 if the device supports a continuous range, or the discrete number.
MINIMUM_CHAR_ EXP_FACTOR	Real. The smallest expansion factor the device supports for Font 1.
MAXIMUM_CHAR_ EXP_FACTOR	Real. The largest expansion factor the device supports for Font 1.
NUM_PREDEF_TEXT_ IND	Integer. The number of predefined text style bundles.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Predefined Text Representation

This inquiry returns information about the predefined text representation specified in the input parameters. This information is usually contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.
TEXT_INDEX	Integer. The index of the text bundle.

Output Parameters

FONT	Integer. The text font in the bundle.
PREC	Integer. The test precision in the bundle.
CHAR_EXP_FACTOR	Real. The character expansion factor.
CHAR_SPACE	Real. The text spacing.
COLOR_INDEX	Integer. The bundle's index into the color table.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_72	Text index is invalid.
GKS$_ERROR_74	A representation for the specified text index has not been predefined on this workstation.

Inquire Fill Area Facilities

This inquiry returns information about the device's fill area capabilities. This data is usually stored in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Modified Parameters

NUM_HATCH_STYLE	Integer. On input, this is the maximum number of hatch styles that the function can write to LIST_HATCH_STYLE. On output, this is the number of entries the handler has written to LIST_HATCH_STYLE.

Output Parameters

NUM_FILL_INTSTLYE — Integer. The number of fill styles the device supports.

LIST_FILL_INTSTYLE — Longword bitmask. Your function should set the following bits if the associated fill style is supplied:

Bit	Style
0	GKS$K_INTSTYLE_HOLLOW (0)
1	GKS$K_INTSTYLE_SOLID (1)
2	GKS$K_INTSTYLE_PATTERN (2)
3	GKS$K_INTSTYLE_HATCH (3)

TOTAL_NUM_HATCH_STYLE — Integer. The number of hatch styles your device supports. If this is larger than NUM_HATCH_STYLE, then LIST_HATCH_STYLE was too small to hold all the hatch styles, and some were not written to the array.

LIST_HATCH_STYLE — Array of integers. The hatch styles the device supports.

NUM_PREDEF_FILL_IND — Integer. The number of predefined fill area styles.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Predefined Fill Area Representation

This inquiry returns information about the predefined fill area representation specified in the input parameter. This information is usually contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.
FILL_INDEX	Integer. The index into the fill area table.

Output Parameters

FILL_INTSTYLE	Integer. One of: • GKS$K_INTSTYLE_HOLLOW (0) • GKS$K_INTSTYLE_SOLID (1) • GKS$K_INTSTYLE_PATTERN (2) • GKS$K_INTSTYLE_HATCH (3)
FILL_STYYE_IND	Integer. The style index in the bundle.
COLOR_INDEX	Integer. The index into the color table.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_80	Fill area index is invalid.
GKS$_ERROR_82	A representation for the specified fill area index has not been predefined on this workstation.

Inquire Pattern Facilities

This inquiry returns information about the workstation type's pattern capabilities. This information is usually contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

NUM_PREDEF_PATT_IND	Integer. The number of predefined patterns.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Predefined Pattern Representation

This inquiry returns information about the predefined pattern representation specified in the input parameters. This information is usually contained in the WDT.

Required

Required for all workstations.

Input Parameters

WSTYPE	Integer. The workstation type.
PATT_INDEX	Integer. The pattern bundle index.
COL_MAJOR_FLAG	Integer flag. GKS$K_TRUE (1) means the array is column major, and GKS$K_FALSE (0) means it is row major.

Modified Parameters

NUM_ROWS	Integer. On input, this is the maximum number of rows that the function can write to PATT_ARRAY. On output, this is the number of rows the handler has written to PATT_ARRAY.
NUM_COLS	Integer. On input, this is the maximum number of columns that the function can write to PATT_ARRAY. On output, this is the number of rows the handler has written to PATT_ARRAY.

Output Parameters

PATT_ARRAY	Two-dimensional array of integers. The index into the color table associated with each cell in the array.
PATT_DIM_X	Integer. The total number of rows in the pattern. If this is larger than NUM_ROWS, then PATT_ARRAY was too small to hold all the rows in the pattern, and some were not written to the array.
PATT_DIM_Y	Integer. The total number of columns in the pattern. If this is larger than NUM_COLS, then PATT_ARRAY was too small to hold all the columns in the pattern, and some were not written to the array.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_85	Specified pattern index is invalid.
GKS$_ERROR_89	A representation for the specified pattern index has not been defined for this workstation.
GKS$_ERROR_90	Interior style PATTERN is not supported on this workstation.

Inquire Color Facilities

This function returns information about the workstation type's color capabilities. This information is normally contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

NUM_COLORS	Integer. The number of colors the device supports.
COLOR_AVAILABLE	Integer flag. GKS$K_MONOCHROME (0) means the device is monochrome and does not support color. GKS$K_COLOR (1) means the device supports color.
NUM_PREDEF_ COLOR_REP	Integer. The number of defined colors.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Predefined Color Representation

This inquiry returns information about the predefined color representation. This information is usually contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.
COLOR_INDEX	Integer. The index into the color table.

Output Parameters

RED	Real. The red component of the color.
GREEN	Real. The green component of the color.
BLUE	Real. The blue component of the color.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_93	Color index is invalid.
GKS$_ERROR_95	A representation for the specified color index has not been predefined on this workstation.

Inquire GDP Primitives

This inquiry returns information about the GDP capabilities of the device you specify in the input parameters. This information is usually contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Modified Parameters

NUM_GDP	Integer. On input, this is the maximum number of GDPs that the function can write to LIST_GDP. On output, this is the number of entries the handler has written to LIST_GDP.

Output Parameters

LIST_GDP	Array of integers. The GDP identifiers.
TOTAL_GDP	Integer. The number of GDPs your device supports. If this is larger than NUM_GDP, then LIST_GDP was too small to hold all the GDPs, and some were not written to the array.
TOTAL_GDP	Integer. The total number of GDPs the device supports.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Generalized Drawing Primitive

This inquiry returns information about the GDP in the input parameters. This information is usually contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.
GDP_ID	Integer. The GDP index.

Output Parameters

GDP_ATTRIBUTES — 32-bit bitmask that represents the attributes used in the GDP. If the bit is set, the corresponding attributes are used. The following bits are defined:

Bit	Attribute
0	GKS$K_POLYLN_ATTRI (0)
1	GKS$K_POLYMR_ATTRI (1)
2	GKS$K_TEXT_ATTRI (2)
3	GKS$K_FILLAR_ATTRI (3)

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_41	Specified workstation type is not able to generate the specified generalized drawing primitive.

Inquire Maximum Length of Workstation State Tables

This inquiry returns the maximum length of each type of bundle table in the workstation state list for the workstation type you specify. This information is normally contained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

MAX_PLINE_BUNDLES	Integer. The maximum number of the polyline bundle table.
MAX_PMARK_ BUNDLES	Integer. The maximum number of polymarker bundle table entries.
MAX_TEXT_BUNDLES	Integer. e maximum number of text bundle table entries.
MAX_FILL_BUNDLES	Integer. The maximum number of fill bundle table entries.
MAX_PATT_IND	Integer. The maximum number of pattern bundle table entries.
MAX_COLOR_IND	Integer. The maximum number of color bundle table entries.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Number of Available Logical Input Devices

This inquiry returns the number of logical input devices of each type available on the workstation type. This information is normally contained in the WDT.

If the device does not support a particular logical input device type, it should return the value 0.

Required

Required for INPUT and OUTIN workstations that support STROKE input.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

NUM_LOC_DEV	Integer. The number of locator logical input devices.
NUM_STK_DEV	Integer. The number of stroke logical input devices.
NUM_VAL_DEV	Integer. The number of valuator logical input devices.
NUM_CHOICE_DEV	Integer. The number of choice logical input devices.
NUM_PICK_DEV	Integer. The number of pick logical input devices.
NUM_STRING_DEV	Integer. The number of string logical input devices.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Default Locator Device Data

This inquiry returns default information about the locator input device specified in the input parameters. This information is normally contained in the WDT.

Required

Required for OUTIN workstation and INPUT workstations that support LOCATOR input.

Input Parameters

WSTYPE	Integer. The workstation type.
DEVNUM	Integer. The locator logical input device number.

Modified Parameters

NUMBER_PET	Integer. On input, this is the maximum number of PETs that the function can write to LIST_PROMPT_ECHO_TYPES. On output this is the number of PETs the handler has written to LIST_PROMPT_ECHO_TYPES.
DATA_REC_SIZE	Integer. On input, this is the number of bytes that the function can write to LOC_DATA_RECORD. On output, this is the number of bytes the handler has written to LOC_DATA_RECORD.

Output Parameters

INIT_LOCN_X	Real. The X coordinate of the initial locator position, in NDC.
INIT_LOCN_Y	Real. The Y coordinate of the initial locator position, in NDC.
NUM_PROMPT_ ECHO_TYPES	Integer. The total number of PETs defined for this workstation and input type. If this is greater than NUMBER_PET, then LIST_PROMPT_ECHO_TYPES was too small to hold all the PETs, and some were not written to the array.
LIST_PROMPT_ECHO_ TYPES	Array of integers. The available prompt and echo types.
ECHO_AREA	Array of four real numbers that describe the echo area in LDC, in the order XMIN, XMAX, YMIN, YMAX.
LOC_DATA_RECORD	Array. The default locator data record.
LOC_DATA_REC_SIZE	Integer. The total size of the locator data record. If this is larger than DATA_REC_SIZE, then LOC_DATA_ RECORD was too small to contain the entire data record, and does not contain the entire data record.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Inquire Default Stroke Device Data

This inquiry returns default information about the stroke input device specified in the input parameters. This data is normally contained in the WDT.

Required

Required for OUTIN workstations and INPUT workstations that support STROKE input.

Input Parameters

WSTYPE	Integer. The workstation type.
DEVNUM	Integer. The stroke logical input device.

Modified Parameters

NUMBER_PET	Integer. On input, this is the maximum number of PETs that the function can write to LIST_PROMPT_ECHO_TYPES. On output this is the number of PETs the handler has written to LIST_PROMPT_ECHO_TYPES.
DATA_REC_SIZE	Integer. On input, this is the number of bytes that the function can write to STROKE_DATA_RECORD. On output, this is the number of bytes the handler has written to STROKE_DATA_RECORD.

Output Parameters

MAXIMUM_BUFSIZE	Integer. The stroke input buffer size.
NUM_PROMPT_ ECHO_TYPES	Integer. The total number of PETs defined for this workstation and input type. If this is greater than NUMBER_PET, then LIST_PROMPT_ECHO_TYPES was too small to hold all the PETs, and some were not written to the array.
LIST_PROMPT_ECHO_ TYPES	Array of integers. The prompt and echo types the device supports.
ECHO_AREA	Array of four real numbers that define the echo area in LDC, in the order XMIN, XMAX, YMIN, YMAX.
STROKE_DATA_ RECORD	Array. The default stroke data record.
TOTAL_DATA_REC_ SIZE	Integer. The total size of the stroke data record. If this is larger than DATA_REC_SIZE, then STROKE_DATA_ RECORD was not large enough to contain the entire data record, and does not contain the entire data record.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Inquire Default Valuator Device Data

This inquiry returns default information about the valuator input device on the workstation type you specify in the input parameters. This information is normally contained in the WDT.

Required

Required for OUTIN workstations and INPUT workstations that support VALUATOR input.

Input Parameters

WSTYPE	Integer. The workstation type.
DEVNUM	Integer. The valuator logical device number.

Modified Parameters

NUMBER_PET	Integer. On input, this is the maximum number of PETs that the function can write to LIST_PROMPT_ECHO_TYPES. On output this is the number of PETs the handler has written to LIST_PROMPT_ECHO_TYPES.
DATA_REC_SIZE	Integer. On input, this is the number of bytes that the function can write to VAL_DATA_RECORD. On output, this is the number of bytes the handler has written to VAL_DATA_RECORD.

Output Parameters

INIT_VALUE	Real. The initial valuator value.
NUM_PROMPT_ ECHO_TYPES	Integer. The total number of PETs defined for this workstation and input type. If this is greater than NUMBER_PET, then LIST_PROMPT_ECHO_TYPES was too small to hold all the PETs, and some were not written to the array.
LIST_PROMPT_ECHO_ TYPES	Array of integers. The prompt and echo types the device supports.
ECHO_AREA	Array of four real numbers that define the echo area in LDC, in the order XMIN, XMAX, YMIN, YMAX.
VAL_DATA_RECORD	Array. The default data record.
TOTAL_DATA_REC_ SIZE	Integer. The total size of the valuator data record. If this is larger than DATA_REC_SIZE, then VAL_DATA_ RECORD was too small to hold the entire data record, and does not contain the entire data record.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Inquire Default Choice Device Data

This inquiry returns default information about the choice input device you specify in the input parameters. This information is normally contained in the WDT.

Required

Required for OUTIN workstations and INPUT workstations that support CHOICE input.

Input Parameters

WSTYPE	Integer. The workstation type.
DEVNUM	Integer. The choice logical input device.

Modified Parameters

NUMBER_PET	Integer. On input, this is the maximum number of PETs that the function can write to LIST_PROMPT_ECHO_TYPES. On output this is the number of PETs the handler has written to LIST_PROMPT_ECHO_TYPES.
DATA_REC_SIZE	Integer. On input, this is the number of bytes that the function can write to CHOICE_DATA_RECORD. On output, this is the number of bytes the handler has written to CHOICE_DATA_RECORD.

Output Parameters

MAX_NUM_CHOICE	Integer. The maximum number of choices.
NUM_PROMPT_ ECHO_TYPES	Integer. The total number of PETs defined for this workstation and input type. If this is greater than NUMBER_PET, then LIST_PROMPT_ECHO_TYPES was too small to hold all the PETs, and some were not written to the array.
LIST_PROMPT_ECHO_ TYPES	Array of integers. The prompt and echo types the device supports.
ECHO_AREA	Array of four real numbers that describe the echo area in LDC, in the order XMIN, XMAX, YMIN, YMAX.
CHOICE_DATA_ RECORD	Array. The default choice data record.
TOTAL_DATA_REC_ SIZE	Integer. The size of the default data record. If this is larger than DATA_REC_SIZE, CHOICE_DATA_RECORD was not large enough to contain the entire data record, and does not contain the entire data record.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Inquire Default String Device Data

This inquiry returns default information about the string logical input device specified in the input parameters.

Required

Required for OUTIN workstations and INPUT workstations that support STRING input.

Input Parameters

WSTYPE	Integer. The workstation type.
DEVNUM	Integer. The string logical input device.

Modified Parameters

NUMBER_PET	Integer. On input, this is the maximum number of PETs that the function can write to LIST_PROMPT_ECHO_TYPES. On output this is the number of PETs the handler has written to LIST_PROMPT_ECHO_TYPES.
DATA_REC_SIZE	Integer. On input, this is the number of bytes that the function can write to STRING_DATA_RECORD. On output, this is the number of bytes the handler has written to STRING_DATA_RECORD.

Output Parameters

MAX_BUFSIZE	Integer. The maximum size of the STRING input buffer.
NUM_PROMPT_ ECHO_TYPES	Integer. The total number of PETs defined for this workstation and input type. If this is greater than NUMBER_PET, then LIST_PROMPT_ECHO_TYPES was too small to hold all the PETs, and some were not written to the array.
LIST_PROMPT_ECHO_ TYPES	Array of integers. The prompt and echo types the device supports.
ECHO_AREA	Array of four real numbers describing the echo area in LDC, in the order XMIN, XMAX, YMIN, YMAX
STRING_DATA_ RECORD	Array. The default string data record.
TOTAL_DATA_REC_ SIZE	Integer. The total size of the default data record. If this is larger than DATA_REC_SIZE, STRING_DATA_RECORD was not large enough to contain the entire data record, and does not contain the entire data record.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$K_ERROR_140	Specified input device is not present on this workstation.

Inquire Default Pick Device Data

This inquiry returns default information about the pick input device you specify. This information is normally maintained in the WDT.

Required

Required for OUTIN workstations and INPUT workstations that support PICK input.

Input Parameters

WSTYPE	Integer. The workstation type.
DEVNUM	Integer. The pick logical device number.

Modified Parameters

NUMBER_PET	Integer. On input, this is the maximum number of PETs that the function can write to LIST_PROMPT_ECHO_TYPES. On output this is the number of PETs the handler has written to LIST_PROMPT_ECHO_TYPES.
DATA_REC_SIZE	Integer. On input, this is the number of bytes that the function can write to PICK_DATA_RECORD. On output, this is the number of bytes the handler has written to PICK_DATA_RECORD.

Output Parameters

NUM_PROMPT_ ECHO_TYPES	Integer. The total number of PETs defined for this workstation and input type. If this is greater than NUMBER_PET, then LIST_PROMPT_ECHO_TYPES was too small to hold all the PETs, and some were not written to the array.
LIST_PROMPT_ECHO_ TYPES	Array of integers. The list of available PETs.
ECHO_AREA	Array of four real numbers that describe the echo area in LDC, in the order XMIN, XMAX, YMIN, YMAX.
PICK_DATA_RECORD	Array. The default pick data record.
TOTAL_DATA_REC_ SIZE	The total size of the pick data record. If this is larger than DATA_REC_SIZE, then PICK_DATA_RECORD was too small to contain the entire data record, and does not contain the entire data record.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_140	Specified input device is not present on this workstation.

Inquire Dynamic Modification of Workstation Attributes

This inquiry returns flags that indicate whether workstations of the type specified can dynamically modify certain attributes. This information is usually maintained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE — Integer. The workstation type.

Output Parameters

DMAF_POLYLINE — Integer. A flag stating whether polyline bundle representations are dynamically changeable. Either:

- GKS$K_IRG (0) Implicit regeneration necessary
- GKS$K_IMM (1) Performed immediately

DMAF_POLYMARKER — Integer. A flag stating whether polymarker bundle representations are dynamically changeable. Either:

- GKS$K_IRG (0) Implicit regeneration necessary
- GKS$K_IMM (1) Performed immediately

DMAF_TEXT — Integer. A flag stating whether text bundle representations are dynamically changeable. Either:

- GKS$K_IRG (0) Implicit regeneration necessary
- GKS$K_IMM (1) Performed immediately

DMAF_FILL — Integer. A flag stating whether fill bundle representations are dynamically changeable. Either:

- GKS$K_IRG (0) Implicit regeneration necessary
- GKS$K_IMM (1) Performed immediately

DMAF_PATTERN — Integer. A flag stating whether pattern bundle representations are dynamically changeable. Either:

- GKS$K_IRG (0) Implicit regeneration necessary
- GKS$K_IMM (1) Performed immediately

DMAF_COLOR — Integer. A flag stating whether color bundle representations are dynamically changeable. Either:

- GKS$K_IRG (0) Implicit regeneration necessary
- GKS$K_IMM (1) Performed immediately

DMAF_TRANSFORM — Integer. A flag stating whether the workstation transformation is dynamically changeable. Either:

- GKS$K_IRG (0) Implicit regeneration necessary
- GKS$K_IMM (1) Performed immediately

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Default Deferral State Values

This inquiry returns the default deferral mode and implicit regeneration mode values for the specified workstation type. This data is usually maintained in the WDT.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

DEF_MODE — Integer. The default deferral mode. One of:

- GKS$K_ASAP (0)
- GKS$K_BNIG (1)
- GKS$K_BNIL (2)
- GKS$K_ASTI (3)

REGEN_MODE — Integer. The default implicit regeneration mode. One of:

- GKS$K_IRG_SUPPRESSED (0)
- GKS$K_IRG_ALLOWED (1)

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Number of Segment Priorities Supported

This inquiry returns the number of segment priorities supported by the workstation type you specify. This information is normally stored in the WDT.

Required

Required for OUTPUT and OUTIN workstations that suppport segments.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

NUM_SEGMENT_ PRIORITIES	Integer. The number of segment priorities supported on this device. This is zero if the device supports an infinite range of priorities, otherwise it is the discrete number.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Dynamic Modification of Segment Attributes

This inquiry returns flags that indicate the ability of the specified workstation type to perform dynamic modification of segment attributes. This information is normally stored in the WDT.

Note that if the handler does not support segments, only the attribute DMAF_HIGHLIGHTING is meaningful.

Required

Required for OUTPUT and OUTIN workstations.

Input Parameters

WSTYPE	Integer. The workstation type.

Output Parameters

DMAF_SEGMENT	Integer. A flag indicating whether the segment transformation can be changed dynamically. Either: • GKS$K_IRG (0) Implicit regeneration necessary • GKS$K_IMM (1) Performed immediately
DMAF_INVISIBILITY	Integer. A flag indicating whether segments can be made invisible dynamically. Either: • GKS$K_IRG (0) Implicit regeneration necessary • GKS$K_IMM (1) Performed immediately

DMAF_VISIBILITY	Integer. A flag indicating whether segments can be made visible dynamically. Either: • GKS$K_IRG (0) Implicit regeneration necessary • GKS$K_IMM (1) Performed immediately
DMAF_HIGHLIGHTING	Integer. A flag indicating whether segments can be highlighted dynamically. Either: • GKS$K_IRG (0) Implicit regeneration necessary • GKS$K_IMM (1) Performed immediately
DMAF_SEGMENT_ PRIORITY	Integer. A flag indicating whether segment priorities can be changed dynamically. Either: • GKS$K_IRG (0) Implicit regeneration necessary • GKS$K_IMM (1) Performed immediately
DMAF_SEGMENT_ OVERLAP	Integer. A flag indicating whether segment overlapping can be changed dynamically. Either: • GKS$K_IRG (0) Implicit regeneration necessary • GKS$K_IMM (1) Performed immediately
DMAF_DELETE_ SEGMENT	Integer. A flag indicating whether segments can deleted dynamically. Either: • GKS$K_IRG (0) Implicit regeneration necessary • GKS$K_IMM (1) Performed immediately

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Inquire Size of Handler Storage

This inquiry returns the amount of storage that the handler needs. The kernel allocates this amount of virtual memory for the handler. The address of the storage space is passed to functions that may require the WSL, in the parameter WSL. This function is only called before a workstation is opened, so the size of the storage area is fixed and cannot be changed.

Required

Required for all workstations.

Input Parameters

None.

Output Parameters

STORAGE_SIZE	Integer. The storage size needed, in bytes.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Chapter 7

Workstation Handler Metafile Functions

This chapter describes the workstation handler metafile functions. You must provide the function WRITE ITEM TO METAFILE if your device is type MO. You must provide the functions GET ITEM TYPE FROM METAFILE and READ ITEM FROM METAFILE if your device is type MI.

The kernel calls the function WRITE ITEM TO METAFILE when the user calls the GKS$WRITE_ITEM function. The function writes a user-defined data type to the metafile.

The kernel calls the GET ITEM TYPE FROM METAFILE function when the user or application calls GKS$GET_ITEM. This function reads an item from the metafile, determines the item type, and calculates the item length. Note that although your metafiles can contain information in any format, DEC GKS can only interpret metafile items according to the format specified in the ANSI GKS standard, with data items structured as defined in Table 7–1, Required Metafile Format. This format is also described in the *DEC GKS Reference Manual*. Your GET ITEM TYPE FROM METAFILE function must translate the data item to this format and return the length of the translated item.

Table 7–1: Required Metafile Format

Item	Required Value	Definition
V	1	Version number.
H	4	Number of characters in the string GKSM displayed at the beginning of each record.
T	3	Length of item type indicator field.
L	8	Length of item data record length indicator field.
I	8	Length of field for each integer in the item data record.
R	14	Length of field for each real in the item data record.
F	1	All numbers formatted according to ISO standard 6093.
RI	1	Reals are not scaled integers.

The remaining header items listed in the GKS standard must be determined by your function, listed in Table 7–2, Metafile Format Items Determined by Your Functions.

Table 7–2: Metafile Format Items Determined by Your Functions

Item	Definition
GKSM	The string GKSM.
N	The name of the author/installation.
D	Date, in the format yy/mm/dd.
ZERO	Not required.
ONE	Not required.

All values should be right-justified and blank- or space-padded in their fields.

The kernel calls READ ITEM FROM METAFILE when the user or application gives the command GKS$READ_ITEM. Since a call to READ ITEM FROM METAFILE must always be preceded by a call to GET ITEM TYPE FROM METAFILE, you may let your GET ITEM TYPE function store the translated item in the local data area. If so, your READ ITEM function need only pass the address of the translated item, not translate the record again.

Write Item to Metafile

This function writes a data record, containing nongraphical data provided by the applications program, to the GKS metafile. The GKS kernel calls it in response to the GKS$WRITE_ITEM call from the application.

The kernel uses this function only to write an implementation-specific data type to the metafile. These data types always have item types greater than 100. The function should add whatever any information your metafiles require to the beginning of the data passed in the input parameters, then write the header and data string to the metafile. Note that if you plan to read the data back using DEC GKS, your metafile input functions must be able to interpret the header.

Required

Required for MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
ITEM_TYPE	Integer. The user-specified data type. This must be greater than 101.
ITEM_DATA_ RECORD_LENGTH	Integer. The length in bytes of the data record.
ITEM_DATA_RECORD	Address. Record of length specified in the previous parameter.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_160	Item type is not allowed for user items.
GKS$_ERROR_161	Item length is invalid.

Get Item from Metafile

This function gets the item type and the length of the current item in the metafile. The length must be the length of the item after it is converted to the format specified at the beginning of this chapter, so this function must convert the record. The item type must be a type specified in the GKS standard, or a user item type. If these conditions are not, met the GKS kernel will be unable to interpret the item.

Note that in most cases a call to READ ITEM FROM METAFILE follows this function. You may choose to write the converted record to the local data area so that the READ ITEM FROM METAFILE function does not need to convert it again.

Required

Required for MI workstations.

Input Parameters

WSL	The address of the handler's local data area.

Output Parameters

ITEM_TYPE	Integer. The current record's item type.
ITEM_DATA_ RECORD_LENGTH	Integer. The length, in bytes, of the current record, after conversion to the format DEC GKS requires.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_162	No item is left in GKS metafile input.
GKS$_ERROR_163	Metafile item is invalid.

Read Item from Metafile

This function returns the current data record, converted to the format DEC GKS requires, then marks the next data record as the new current data record. If the item is larger than the MAXIMUM_ITEM_DATA_RECORD_LENGTH, the excess is truncated. If MAXIMUM_ITEM_DATA_RECORD_LENGTH is 0, the function returns no data record, and marks the next data record as the new current data record.

Note that if your GET ITEM TYPE FROM METAFILE function stores the translated record, this function need only find and return the converted record.

Required

Required for MI workstations.

Input Parameters

WSL	The address of the handler's local data area.
MAXIMUM_ITEM_ DATA_RECORD_ LENGTH	Integer. Size of the buffer to which the record should be written. If 0, no record should be written. If the item data record is longer than the buffer, truncate the record.

Output Parameters

DATA_RECORD	Address. The metafile data record, passed by reference.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_162	No item is left in GKS metafile output.
GKS$_ERROR_163	Metafile item is invalid.
GKS$_ERROR_165	Content of item data record is invalid for the specified item type.
GKS$_ERROR_166	Maximum item data record length is invalid.

Chapter 8

Workstation Handler Set Representation Functions

This chapter describes the functions that set output representations. Output representations control the appearance of the output your device draws, such as the style, color, and width of polylines. These functions are required only for workstations of type OUTPUT, OUTIN, and MO.

When the function Open Workstation initializes an OUTPUT or OUTIN workstation, it should load the predefined bundles for each attribute into the workstation's WSL. The functions described in this chapter replace predefined bundles, or define output styles in addition to the predefined bundles.

The changes these functions make remain in effect until the user changes them again, or until the user closes the workstation.

If your device cannot implement the exact values passed to it (for example, if the device cannot use the exact color intensity or linewidth scale factor passed as an input parameter), the device should use the closest approximation that it can accept. How you implement this (for example, by rounding up or down) is implementation dependent. However, your handler must store the set value (the value passed to it) as well as the realized value (the value it actually uses), and be able to return these values in response to inquiry functions.

Note that these functions update the WSL. If the handler is providing segment support, and there are primitives that take attributes from the bundle representation that was changed, then the displayed image must be updated as follows:

- If the value of the corresponding dynamic modification flag is GKS$K_IMM, then the change to the image (to reflect the new bundle values) must be made immediately. These changes may affect primitives outside of segments. Primitives outside of segments must be maintained on the display surface.

- If the value of the corresponding dynamic modification flag is GKS$K_IRG, then a regeneration of the image is needed. In this case, if the value of the REGEN_MODE flag (normally maintained in the WSL) is GKS$K_IRG_ALLOWED, then a regeneration is performed. Otherwise, regeneration is suppressed and the NEWFRAME_ACTION_NECESSARY_AT_UPDATE flag is set to GKS$K_NEWFRAME_NECESSARY.

8.1 Function Descriptions

This section lists the Set Representation function descriptions.

Set Polyline Representation

This function sets a new polyline bundle for the device. The bundle is usually stored in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL — The address of the handler's local data area.

PLINE_INDEX — Integer. The polyline bundle to be set.

LINE_TYPE — Integer. The GKS Standard defines the following linetypes.

Description	Linetype	DEC GKS Constant
Solid	1	GKS$K_LINE_TYPE_SOLID (1)
Dashed	2	GKS$K_LINE_TYPE_DASHED (2)
Dotted	3	GKS$K_LINE_TYPE_DOTTED (3)
Dashed-Dotted	4	GKS$K_LINE_TYPE_DASHED_DOTTED (4)

Linetypes less than 0 are implementation defined. Linetypes greater than 4 are reserved for registration or future standardization.

LINEWIDTH_SCALE_FACTOR — Real. The factor which, multiplied by the nominal linewidth, yields the desired linewidth.

COLOR_INDEX — Integer. The index into the color table. This controls the color the line will appear in.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_60	Polyline index is invalid.
GKS$_ERROR_64	Specified linetype is not supported on this workstation.
GKS$_ERROR_93	Color index is invalid.

Set Polymarker Representation

This function sets a new polymarker bundle for the device. The bundle is usually stored in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL — The address of the handler's local data area.

PMARK_INDEX — Integer. The polymarker bundle to be set.

MARKER_TYPE — Integer. The markertype. The GKS Standard defines the following markertypes:

Description	Marker-type	DEC GKS Constant
Dot (.)	1	GKS$K_MARKERTYPE_DOT (1)
Plus sign (+)	2	GKS$K_MARKERTYPE_PLUS (2)
Asterisk (*)	3	GKS$K_MARKERTYPE_ASTERISK (3)
Circle (O)	4	GKS$K_MARKERTYPE_CIRCLE (4)
Diagonal Cross (X)	5	GKS$K_MARKERTYPE_DIAGONAL_CROSS (5)

Markertypes less than 0 are implementation defined. Markertypes greater than 5 are reserved for registration or future standardization.

MSIZE_SCALE_FACTOR — Real. The factor that, when multiplied by the nominal markersize, yields the desired markersize.

COLOR_INDEX	Integer. The index into the color table. This controls the color the marker will appear in.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_61	Polymarker index is invalid.
GKS$_ERROR_70	Specified markertype is not supported on this workstation.
GKS$_ERROR_93	Color index is invalid.

Set Text Representation

This function sets a new text bundle for the device. Bundles are usually stored in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
TEXT_INDEX	Integer. The text representation to be set.
FONT	Integer. The font to be used.
PREC	Integer. The precision.
CHAR_EXP_FACTOR	Real. The factor which, when mulitipled by the nominal character size, yields the desired character size.
CHAR_SPACE	Real. Amount (in LDC) of additional space placed between consecutive characters.
COLOR_INDEX	Integer. The index into the color table. This controls the color the text will appear in.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_72	Text index is invalid.
GKS$_ERROR_76	Requested text font is not supported for the specified precision on this workstation.
GKS$_ERROR_93	Color index is invalid.

Set Fill Area Representation

This function sets a new fill area bundle for the device. Bundles are usually stored in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL — The address of the handler's local data area.

FILL_INDEX — Integer. The index to be set.

FILL_INTSTYLE — Integer. The interior style. The fill area interior style must be one of the following:

Pattern	Integer Value	DEC GKS Constant
Hollow	0	GKS$K_INTSTYLE_HOLLOW
Solid	1	GKS$K_INTSTYLE_SOLID
Pattern	2	GKS$K_INTSTYLE_PATTERN
Hatched	3	GKS$K_INTSTYLE_HATCH

FILL_STYLE_INDEX — Integer. The contents of this variable depend on the FILL_INTSTYLE value, as shown in the following table:

Style	Meaning
Hollow	Unused.
Solid	Unused.
Pattern	Index into the workstation's pattern table. Must be greater than 0.
Hatched	Values less than zero indicate implementation-specific hatch patterns. Values greater than 0 are reserved for registration or future standardization.

COLOR_INDEX — Integer. The index into the color table. This controls the color the fill area will appear in.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_80	Fill area index is invalid.
GKS$_ERROR_83	Specified fill area interior style is not supported on this workstation.
GKS$_ERROR_85	Specified pattern index is invalid.
GKS$_ERROR_86	Specified hatch style is not supported on this workstation.
GKS$_ERROR_93	Color index is invalid.

Set Pattern Representation

This function sets a pattern bundle for the device. The input parameters specify a bundle index and define an array of color indexes. Each element in the array corresponds to a cell in the pattern, and its value is an index into the color table. When the handler outputs this pattern, it colors each cell according to the indexes in the pattern array.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
PATTERN_INDEX	Integer. The pattern index to be set.
PATT_DIM_X	Integer. The number of rows in the pattern.
PATT_DIM_Y	Integer. The number of columns in the pattern.
PATT_ARRAY	Integer array. This is a two-dimensional array, passed by reference, in which each integer represents one cell of the pattern.
COL_MAJOR_FLAG	Integer flag. GKS$K_TRUE means the array is in column-major format. GKS$K_FALSE means it is in row-major format.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_85	Specified pattern index is invalid.
GKS$_ERROR_90	Interior style PATTERN is not supported on this workstation.
GKS$_ERROR_93	Color index is invalid.

Set Color Representation

This function sets a color bundle for the device. Bundles are usually stored in the WSL.

Required

Required for OUTPUT, OUTIN, and MO workstations.

Input Parameters

WSL	The address of the handler's local data area.
COLOR_INDEX	Integer. The color bundle to be set.
RED	Real. The red component of the color.
GREEN	Real. The green component of the color.
BLUE	Real. The blue component of the color.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_93	Color index is invalid.

Chapter 9

Workstation Handler Output Functions

This chapter describes the workstation handler output functions. These are required for OUTPUT, OUTIN, and MO workstations. You must provide an output function for each output type.

This chapter also includes the function HIGHLIGHT EXTENT. You only need to include the HIGHLIGHT EXTENT function if you want the GKS kernel to simulate segments.

9.1 Coordinate Data

Coordinate data controls the location and shape of the output on the display surface. The kernel passes one or more points, in world coordinates, to the output functions. The handler function then converts the world coordinate points to device coordinates according to the current normalization and workstation transformations, and uses these coordinates to position the output.

The kernel passes points to the functions Polyline, Polymarker, Fill Area, and GDP, using three parameters. The first parameter, NUM_PTS, tells the function how many points the kernel will pass. The second parameter, X_WC_ARRAY, is the address of an array containing the X coordinates of the input points, in WC. The third parameter, Y_WC_ARRAY, is the address of the array of Y coordinates in WC.

The two arrays contain a number of elements equal to NUM_PTS. For these functions the kernel also passes the address of two additional arrays, TEMP_X_ARRAY and TEMP_Y_ARRAY. These arrays are also of length NUM_PTS, except for those used in FILL AREA functions, which are (number of points + 1). Your function may use these arrays to temporarily hold the DC coordinates it computes.

For the output function TEXT, the kernel passes the parameters WC_INIT_X and WC_INIT_Y. These parameters contain the X and Y coordinates of the text position, in WC. The handler must convert the coordinates to DC.

For the output function CELL ARRAY, the kernel passes the parameters P_X, P_Y, Q_X, Q_Y, R_X, and R_Y. These parameters contain the X and Y coordinates of three corners of the output parallelogram. All the points are in WC, and your function must convert them to DC.

9.2 Attributes

The kernel passes geometric attributes to the output functions TEXT, FILL AREA, and CELL ARRAY. These attributes are specific to each function. See the function descriptions for information about them.

The kernel also passes nongeometric attributes to the output functions. Nongeometric attributes control the appearance of output. For example, linetype, markersize, and color are all nongeometric attributes.

Attributes are passed in the active attribute array. For information about the active attribute array, see Section 4.1, Active Attribute Array.

Note that if you simulate output functions within your handler by calling other handler functions (for example, if you simulate fill areas by making multiple POLYLINE calls), the calling function must generate and pass the attribute array as well as the other parameters required by the function it calls.

9.3 Aspect Source Flags

The GKS kernel passes a 32-bit bitmask to the output functions. This parameter tells your handler whether to use the individual or bundled nongeometric attributes for each output primitive. The bundled attributes are the attributes contained in the bundle named by the attribute index. For example, POLYLINE_INDEX, passed in the Active Attribute Array, contains an integer that represents one of the workstation's polyline bundles. The individual primitives are the primitives passed in the Active Attribute Array, such as LINE_TYPE and LINEWIDTH_SCALE_FACTOR.

Bits 0 through 12 represent individual attributes. If a given bit is SET, then use the individual value for that attribute. If the bit is clear, use the bundled value for the attribute. Table 9–1 shows which bit corresponds with which attribute.

Table 9–1: ASF Bits

Aspect	Bit	Integer Value
Linetype	0	1
Linewidth	1	2
Polyline color	2	4
Markertype	3	8
Markersize	4	16
Marker color	5	32
Text Font and Precision	6	64
Character Expansion Factor	7	128
Character Space	8	256
Text Color	9	512
Fill Area Pattern	10	1024
Fill Area Interior Style	11	2048
Fill Area Color	12	4096

The ASF bitmask also contains four bits that tell your handler whether the aspects for certain primitives have changed since the last time the primitive was drawn. Your handler should check these "no change" bits if it stores the attributes for each primitive. If it does, and it determines that the attributes have not changed, then it does not need to decipher the rest of the ASF for that primitive.

Table 9–2 lists the no change bits.

Table 9–2: No Change Bits

Primitive	Bit	Integer Value
Polyline	13	8192
Polymarker	14	16384
Text	15	32768
Fill Area	16	65536

Finally, bit 17 (integer value 131072) indicates whether your output function is being called by another handler function, usually in order to simulate the first function. For example, your FILL_AREA function may call your POLYLINE function to simulate fill areas. In this case, the calling function should set bit 17. If your device stores attribute settings, it should recognize that when bit 17

is set, it may need to change some attributes for this primitive, but it should restore the original attributes after this call. As an example, if the last polyline was red, your device may have stored red as the polyline color. If your fill area function calls the polyline function to simulate a blue fill area, it should set bit 17. The polyline function should recognize that because bit 17 is set, it should draw using the color the fill area function passes, but when it finishes drawing, it should restore red as the current color.

The GKS constants file GKSDEFS contains constants for bitmask values. You can use these constants if your handler functions include the GKSDEFS file as described in Chapter 3, Building a Workstation Handler System. Table 9–3 lists the constants.

Table 9–3: GKS Bitmask Constants

Bit	Constant
0	GKS$M_LINETYPE
1	GKS$M_LINEWIDTH
2	GKS$M_PLINE_COLOR
3	GKS$M_MARKERTYPE
4	GKS$M_MARKERSIZE
5	GKS$M_PMARK_COLOR
6	GKS$M_TEXT_FONT_PREC
7	GKS$M_CHAR_EXPAN_FAC
8	GKS$M_CHAR_SPACE
9	GKS$M_TEXT_COLOR
10	GKS$M_FILL_INTER_STYLE
11	GKS$M_FILL_STYLE
12	GKS$M_FILL_COLOR
13	GKS$M_UNCHANGE_PLINE
14	GKS$M_UNCHANGE_PMARK
15	GKS$M_UNCHANGE_TEXT
16	GKS$M_UNCHANGE_FILL
17	GKS$M_SIMULATION

9.4 Segment Overlap

If your handler supports segments, then when an output function adds a primitive to a segment, and that primitive causes the segment to overlap another segment of higher priority, the following occurs:

- If the flag DMAF_SEGMENT_OVERLAP is GKS$K_IMM, then update the image so that the newly added primitive is overlapped by the higher priority segment. Primitives outside the segments are retained.
- Otherwise, if the value of the flag REGEN_MODE is GKS$K_IRG_ALLOWED, then regenerate the image by calling the handler's own REDRAW_ALL_SEGMENTS function. Primitives outside of segments are deleted from the display surface. If the flag REGEN_MODE is GKS$K_IRG_SUPPRESSED, set the NEW_FRAME flag to GKS$K_NEWFRAME_NECESSARY.

9.5 Output Function Descriptions

This section describes the output functions.

Polyline

This function outputs a polyline. A polyline is one or more line segments.

The function performs the following operations:

- Gets the values from the bundle table, if attributes are bundled.
- Converts from WC to DC (if needed).
- Clips against clipping rectangle and workstation window.
- Draws line with given attributes.
- Sets DISPLAY_EMPTY to NOTEMPTY, if a line is drawn on the display.

Required

Required for OUTPUT, OUTIN, and MO devices.

Input Parameters

WSL	The address of the handler's local data area.
NUM_PTS	Integer. The number of points in the line to be drawn.
WC_X_ARRAY	Array of reals of size (NUM_PTS) containing the X component of the points, in WC.
WC_Y_ARRAY	Array of reals of size (NUM_PTS) containing the Y component of the points, in WC.
TEMP_X_ARRAY	Scratch array of size (NUM_PTS) for temporary storage of the DC coordinates.
TEMP_Y_ARRAY	Scratch array of size (NUM_PTS) for temporary storage of the DC coordinates.

ATTRIB_ARRAY — The following elements from the attribute array:

Attribute	Contents
POLYLINE_INDEX	Integer. The index into the polyline bundle table.
LINE_TYPE	Integer. The linetype. One of: • GKS$K_LINETYPE_SOLID (1) • GKS$K_LINETYPE_DASHED (2) • GKS$K_LINETYPE_DOTTED (3) • GKS$K_LINETYPE_DASHED_DOTTED (4)
LINEWIDTH	Real. The linewidth scale factor.
COLOR_INDEX	Integer. The index into the color table.
PICK_ID	Integer. The current pick id.

ASF_MASK — The aspect source flags, as described in Section 9.3, Aspect Source Flags.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Polymarker

This function outputs a polymarker at each point specified in the input parameters. It performs the following operations:

- Gets the values from the bundle table, if attributes are bundled.
- Converts from WC to DC (if needed).
- Clips against clipping rectangle and workstation window. If a point in the list of points is outside the clipping rectangle, then no portion of the corresponding polymarker may be visible. Otherwise, clipping of markers is workstation dependent.
- Draws marker with given attributes.
- Sets DISPLAY_EMPTY to NOTEMPTY, if any markers are drawn.

Required

Required for OUTPUT, OUTIN, and MO devices.

Input Parameters

WSL	The address of the handler's local data area.
NUM_PTS	Integer. The number of markers to be drawn. This must be at least one.
WC_X_ARRAY	Array of reals, with size (NUM_PTS). The X components of the marker coordinates, in WC.
WC_Y_ARRAY	Array of reals, with size (NUM_PTS). The Y components of the marker coordinates, in WC.
TEMP_X_ARRAY	Scratch array of size (NUM_PTS) for temporary storage of the DC coordinates, if needed.
TEMP_Y_ARRAY	Scratch array of size (NUM_PTS) for temporary storage of the DC coordinates, if needed.

ATTRIB_ARRAY — The following attributes from the attribute array:

Attribute	Contents
POLYMARKER_INDEX	Integer. The index into the polymarker bundle table.
MARKER_TYPE	Integer. The polymarker type. One of: • GKS$K_MARKERTYPE_DOT (1) • GKS$K_MARKERTYPE_PLUS (2) • GKS$K_MARKERTYPE_ASTERISK (3) • GKS$K_MARKERTYPE_CIRCLE (4) • GKS$K_MARKERTYPE_DIAGONAL_CROSS (5) Or a workstation-dependent markertype with a value less than 0.
MSIZE_SCALE_FACTOR	Real. The marker width scale factor
POLYMARKER_COLOR_INDEX	Integer. The index into the color table.
PICK_ID	Integer. The current pick id.

ASF_MASK — The aspect source flags, as described in Section 9.3, Aspect Source Flags.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Text

This function outputs a character string. It performs the following operations:

- Gets the values from the bundle table, if attributes are bundled.
- Converts from WC to DC (if needed).
- Clips against clipping rectangle and workstation window. If a portion of the text is outside the clipping rectangle, then no part of the text may be visible. Otherwise, clipping of markers is workstation dependent.
- Draws text with given attributes.
- Sets DISPLAY_EMPTY to NOTEMPTY, if any text is drawn.

If the transformation produces a character height or width of zero, the handler returns an error. In this case, whether or not to produce any output is workstation dependent.

If the character string contains a control character, the workstation may either ignore the character, generate some visual effect, or return an error code.

If clipping is enabled, the device must clip at the clipping rectangle. The portion of the string that falls outside the rectangle is not displayed. However, whether the device draws characters that fall on the clipping rectangle border, or the portion of the string that falls within the rectangle, depends on the text precision.

- For stroke precision text, the device must draw the portion of the string that falls within the clipping rectangle. If a character falls on the clipping rectangle border, the device must output the portion of the character within the rectangle, but not the portion of the character outside of the rectangle.
- For character precision text, the device must draw the portion of the string that falls within the clipping rectangle. If a character falls on the clipping rectangle border, the device may either display the character, not display the character, or display the portion of the character within the rectangle and clip the portion outside.
- For string precision text, the device may either display the portion of the string that falls inside the clipping rectangle, display the entire string, or display none of the string. If your device displays the portion of the string within the rectangle, it may either display any character that falls on the

clipping rectangle border, display the portion of the character within the rectangle, or not display the character.

Note that DIGITAL provides a simulation routine to perform stroke precision text with DEC GKS fonts. For information about implementing text using the stroke precision simulation functions, see Appendix B, Stroke Text Simulation Routines.

Required

Required for OUTPUT, OUTIN, and MO devices.

Input Parameters

WSL	The address of the handler's local data area.
TEXT_POS_X	Real. The X value of the starting text position, in WC.
TEXT_POS_Y	Real. The Y value of the starting text position, in WC.
CHAR_STRING	The actual text character string, passed by class S descriptor.
ATTRIB_ARRAY	The following elements from the attribute array:

Attribute	Contents
TEXT_INDEX	Integer. The index into the text bundle table.
FONT	Integer. The text font number. This may be a DIGITAL-supported font or a handler-specific font.

Attribute	Contents
PREC	Integer. The text precision. One of: • GKS$K_TEXT_PRECISION_STRING (0) • GKS$K_TEXT_PRECISION_CHAR (1) • GKS$K_TEXT_PRECISION_STROKE (2)
CHAR_EXP_FACTOR	Real. The text character expansion factor.
CHAR_SPACE	Real. The text character spacing.
COLOR_INDEX	Integer. The index into the color table.
CHARACTER_HEIGHT_X	X component of the character height vector, in WC.
CHARACTER_HEIGHT_Y	Y component of the character height vector, in WC.
CHARACTER_WIDTH_X	X component of the character width vector, in WC.
CHARACTER_WIDTH_Y	Y component of the character width vector, in WC.
TEXT_PATH	Integer. The direction of the text string. One of: • GKS$K_TEXT_PATH_RIGHT (0) • GKS$K_TEXT_PATH_LEFT (1) • GKS$K_TEXT_PATH_UP (2) • GKS$K_TEXT_PATH_DOWN (3)

Attribute	Contents
TEXT_ALIGNMENT_HORZ	Integer. The horizontal text alignment. One of: • GKS$K_HALIGN_NORMAL (0) • GKS$K_HALIGN_LEFT (1) • GKS$K_HALIGN_CENTER (2) • GKS$K_HALIGN_RIGHT (3)
TEXT_ALIGNMENT_VERT	Integer. The vertical text alignment. One of: • GKS$K_VALIGN_NORMAL (0) • GKS$K_VALIGN_TOP (1) • GKS$K_VALIGN_CAP (2) • GKS$K_VALIGN_HALF (3) • GKS$K_VALIGN_BASE (4) • GKS$K_VALIGN_BOTTOM (5)
PICK_ID	Integer. The current pick id.

ASF_MASK — The aspect source flags, as described in Section 9.3, Aspect Source Flags.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_101	Invalid code in string.

Fill Area

This function fills the polygon defined by the input points according to the fill area style currently selected. It performs the following operations:

- Gets the values from the bundle table, if attributes are bundled.
- Converts from WC to DC (if needed).
- Clips against clipping rectangle and workstation window.
- Draws the fill area with given attributes.
- Sets DISPLAY_EMPTY to NOTEMPTY, if it draws on the display.

If the fill area interior style is HOLLOW, the function draws the boundary of the polygon. For other interior styles, the function draws the fill area so that regions which share a common edge appear without a gap between them, and without overlap to the extent that can reasonably be achieved. If the polygon is clipped, new boundaries that result from clipping replace the original boundaries.

For interior styles other than hollow, the function fills the fill area. If the fill area is complex, it may contain shapes that are not within the fill area. For example, a donut-shaped fill area contains a hollow center that is not within the fill area's borders and should not be filled in. To determine whether a portion of a fill area should be filled, a point is considered to be within the polygon if a straight line drawn from the point in any direction crosses the boundary of the polygon an odd number of times. Note that tangential contacts with the boundary do not count as a crossing.

If transformations make all the points in the fill area boundary the same, then whether anything is drawn is workstation dependent, and no error results. If two or more sides of the boundary polygon have a line segment in common, whether the line segment is considered part of the bounding polygon is implementation dependent, and no error results.

The polygon passed to the handler is not necessarily closed. That is, the first and last point in the point arrays may be different. In this case, your function should close the area by connecting the first and last points. An additional location is provided in the scratch arrays so that a closed, transformed polygon can be built when necessary.

Required

Required for OUTPUT, OUTIN, and MO devices.

Input Parameters

WSL	The address of the handler's local data area.
NUM_PTS	Integer. The number of points that define the fill area. There are always at least three points.
WC_X_ARRAY	Array of reals, size NUM_PTS. The X component of the coordinates, in WC.
WC_Y_ARRAY	Array of reals, size NUM_PTS. The Y component of the coordinates, in WC.
TEMP_X_ARRAY	Scratch array of size (NUM_PTS +1) for temporary storage of the DC coordinates.
TEMP_Y_ARRAY	Scratch array of size (NUM_PTS +1) for temporary storage of the DC coordinates.
ATTRIB_ARRAY	The following attributes are from the attribute array:

Attribute	Contents
FILL_AREA_INDEX	Integer. The index into the fill area bundle table.
FILL_INTSTYLE	Integer. The fill area interior style. One of: • GKS$K_INTSTYLE_HOLLOW (0) • GKS$K_INTSTYLE_SOLID (1) • GKS$K_INTSTYLE_PATTERN (2) • GKS$K_INTSTYLE_HATCH (3)

Attribute	Contents
FILL_STYLE_IND	Integer. The fill area style index. If FILL_INTSTYLE is GKS$K_INTSTYLE_PATTERN, the style index is an index into the pattern table. If FILL_INTSTYLE GKS$K_INTSTYLE_HATCH, then the style index is the device-dependent hatch style.
FILL_AREA_COLOR_INDEX	Integer. The index into the color table.
PATTERN_REFERENCE_POINT_X	Real. If FILL_INTSTYLE is GKS$K_INTSTYLE_PATTERN, then this is the X value of the pattern reference point, in WC.
PATTERN_REFERENCE_POINT_Y	Real. If FILL_INTSTYLE is GKS$K_INTSTYLE_PATTERN, then this is the Y value of the pattern reference point, in WC. If not, this is undefined.
PATTERN_HEIGHT_X	Real. If FILL_INTSTYLE is GKS$K_INTSTYLE_PATTERN, then this is the X component of the pattern height vector for the pattern array. If not, this is undefined.
PATTERN_HEIGHT_Y	Real. If FILL_INTSTYLE is GKS$K_INTSTYLE_PATTERN, then this is the Y component of the pattern height vector for the pattern array. If not, this is undefined.
PATTERN_WIDTH_X	Real. If FILL_INTSTYLE is GKS$K_INTSTYLE_PATTERN, then this is the X component of the pattern width vector for the pattern array. If not, this is undefined.
PATTERN_WIDTH_Y	Real. If FILL_INTSTYLE is GKS$K_INTSTYLE_PATTERN, then this is the Y component of the pattern width vector for the pattern array. If not, this is undefined.
PICK_ID	Integer. The current pick id.

ASF_MASK — The aspect source flags, as described in Section 9.3, Aspect Source Flags.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Cell Array

This function draws a cell array. It performs the following operations:

- Gets the values from the bundle table, if attributes are bundled.
- Converts from WC to DC (if needed).
- Clips against clipping rectangle and workstation window.
- Draws the cell array with given attributes.
- Sets DISPLAY_EMPTY to NOTEMPTY, if the function draws on the display surface.

A cell array is defined as a rectangle divided into a grid. The corners of the rectangles, and the number of rows and columns in the grid, are passed as input parameters. Your handler should divide the rectangle into the number of rows and columns specified in the input parameters. Each resulting cell must be the same size.

Then your handler must color the grid according to the color index array. This is an integer array with the same number of rows and columns. The array is passed as an input parameter, and each integer in the array is an index into the color table. The first index in the color array (specified by the input variables INIT_ROW and INIT_COL, which serve as offsets into the color array) corresponds to the cell in the P corner of the rectangle. The subsequent points are colored row-by-row if the input parameter COL_MAJOR is GKS$K_FALSE, or column-by-column if COL_MAJOR is GKS$K_TRUE. The number of rows and columns in the cell array, and the number of rows and columns following the offset into the color array, are always equal. On a raster display, each pixel with its center within a cell takes the color of the cell.

The cell array grid is subject to all transformations, and these transformations may convert the grid and the cells to parallelograms. If the transformation places part of a cell outside the clipping area, the cell is clipped at the clipping boundary. If transformations make the corners of a cell coincident or colinear, whether any output appears is workstation dependent, and no error results.

If your handler cannot draw cell arrays, drawing the boundary of the cell array is sufficient to conform with the GKS standard.

Cell Array

Required

Required for OUTPUT, OUTIN or MO devices.

Input Parameters

WSL	The address of the handler's local data area.
CELL_RECTANGLE	Array containing the points of the cell rectangle, defined as follows:
	P_X — Real. The X coordinate of the P corner of the cell rectangle, in WC.
	P_Y — Real. The Y coordinate of the P corner of the cell rectangle, in WC.
	Q_X — Real. The X coordinate of the Q corner of the cell rectangle, in WC.
	Q_Y — Real. The Y coordinate of the Q corner of the cell rectangle in WC.
	R_X — Real. The X coordinate of the R corner (the point associated with the DX,1 cell), in WC.
	R_Y — Real. The Y coordinate of the R corner (the point associated with the DX,1 cell), in WC.
INIT_ROW	Integer. The index to the starting row of the color index array.
INIT_COL	Integer. The index to the starting column of the color index array.
DIM_ROWS	Integer. The total number of rows in color index array.
DIM_COLS	Integer. The total number of columns in color index array.
COLOR_INDEX_ ARRAY	The address of the cell array.
DISPLAY_ROW	Integer. The number of row cells to display.
DISPLAY_COL	The number of column cells to display.

COL_MAJOR_FLAG	Integer. A flag that states whether the color index array is column major. Either: • GKS$K_TRUE (0) if the color index array is column major. • GKS$K_FALSE (1) if the color index array is row major.
SCRATCH_CELL	The address of a scratch area the same size as the cell array.
CELL_ARRAY_PICK_ID	Integer. The current pick id.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

GDP

This function draws the graphic primitive specified by the input parameters. It performs the following operations:

- Gets the values from the bundle table, if attributes are bundled.
- Converts from WC to DC (if needed).
- Clips against clipping rectangle and workstation window. If a point in the list of points is outside the clipping rectangle, your function may either draw the GDP, or not draw it and return the status code GKS$ERROR_105.
- Draws GDPs with given attributes.
- Sets DISPLAY_EMPTY to NOTEMPTY, if any GDPs are drawn.

Required

Required for OUTPUT, OUTIN, and MO devices.

Input Parameters

WSL	The address of the handler's local data area.
NUM_PTS	Integer. The number of points where you want to draw the GDP.
WC_X_ARRAY	Array of size NUM_PTS containing the X points, in WC.
WC_Y_ARRAY	Array of size NUM_PTS containing the Y points, in WC.
TEMP_X_ARRAY	Scratch array of size (NUM_PTS) for temporary storage of the DC coordinates.
TEMP_Y_ARRAY	Scratch array of size (NUM_PTS) for temporary storage of the DC coordinates.
GDP_ID	Integer. This identifies the GDP you want.
GDP_DATA_REC_SIZE	Integer. The number of bytes of the GDP data record.

GDP_DATA_RECORD	The address of the GDP data record. This data record is not touched by the kernel and is passed directly as the user created it.
ATTRIB_ARRAY	The entire attribute array defined in Section 4.1, Active Attribute Array.
ASF_MASK	The aspect source flags, as described in Table 9–3, GKS Bitmask Constants.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.
GKS$_ERROR_100	Number of points is invalid.
GKS$_ERROR_102	Generalized drawing primitive identifier is invalid.
GKS$_ERROR_103	Content of generalized drawing primitive data record is invalid.
GKS$_ERROR_104	At least one active workstation is not able to generate the specified generalized drawing primitive.
GKS$_ERROR_105	At least one active workstation is not able to generate the generalized drawing primitive under the current transformation and clipping rectangle.

Highlight Extent

This function highlights the extent specified in the input parameters. The form of highlighting is device dependent. For example, your device could use complement mode to fill the extent, or it could draw a line around the extent using a polyline routine.

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler does not support segments.

Input Parameters

WSL	The address of the handler's local data area.
EXTENT	Array of eight real numbers that define the corners of the extent to be highlighted, in the order X1, Y1, X2, Y2, X3, Y3, X4, Y4.
FLAG	Integer flag. GKS$K_TRUE means highlight the extent. GKS$K_FALSE means remove the highlighting.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Chapter 10

Workstation Handler Segment Functions

This chapter describes the workstation handler segment functions. You must supply these functions if your device supports segments. If your device does not support segments, do not supply these functions, and the GKS kernel will simulate the segment operations. Note that if you let the kernel simulate segments, you must supply the functions HIGHLIGHT EXTENT (described in Chapter 9, Workstation Handler Output Functions) and PERFORM DEFERRED OUTPUT (described in Chapter 4, Workstation Handler Control and Transformation Functions).

Create Segment

This function creates a segment using the specified segment name. The segment name is stored in the set of stored segments in the device's WSL. All output drawn between this call and the next call to CLOSE SEGMENT is collected into this segment.

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.
SEGMENT_NAME	Integer. The name to be associated with the segment.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Close Segment

This function closes the segment that is currently active. Subsequent output primitives are not stored in the segment.

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Rename Segment

This function changes the name of the specified segment to the new name specified.

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.
OLD_SEGMENT_NAME	Integer. The existing segment name.
NEW_SEGMENT_NAME	Integer. The new segment name.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Delete Segment

This function removes the segment name from the set of stored segments. The set of stored segments is usually kept in the WSL. If a segment state list is maintained, this function should also delete it.

If the dynamic modification flag DMAF_DELETE_SEGMENT is GKS$K_IMM (1), then the segment is deleted immediately, but primitives outside of segments remain on the display surface. Otherwise the following occurs:

- If the REGEN_MODE is GKS$K_IRG_ALLOWED, then the image is regenerated. Primitives outside of segments are no longer shown on the display surface.
- If the REGEN_MODE is GKS$K_IRG_SUPPRESSED, the regeneration is suppressed and NEW_FRAME is set to GKS$K_NEWFRAME_NECESSARY.

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.
SEGMENT_NAME	Integer. The name of the segment to be deleted.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set Segment Transformation

This function applies the transformation array to the segment named in the input parameters.

The transformation matrix is passed as an array of six real numbers, ordered as M(1,1), M(1,2), M(1,3), M(2,1), M(2,2), M(2,3). Your handler should arrange them into a matrix of the following form:

$$\begin{bmatrix} m(1,1) & m(1,2) & m(1,3) \\ m(2,1) & m(2,2) & m(2,3) \end{bmatrix}$$

Locations M(1,3) and M(2,3) are in NDC. The other components are unitless. For more information about transformations, see Appendix A, Transformations.

If the dynamic modification flag DMAF_SEGMENT_XFORM is GKS$K_IMM (1), then the transformation is changed immediately, but primitives outside of segments remain on the display surface. Otherwise the following occurs:

- If the REGEN_MODE is GKS$K_IRG_ALLOWED, then the image is regenerated. Primitives outside of segments are no longer shown on the display surface.
- If the REGEN_MODE is GKS$K_IRG_SUPPRESSED, the regeneration is suppressed and NEW_FRAME is set to GKS$K_NEWFRAME_NECESSARY.

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Set Segment Transformation

Input Parameters

WSL	The address of the handler's local data area.
SEGMENT_NAME	Integer. The name of the segment.
TRANSFORMATION_ ARRAY	The address of an array of six real numbers. The transformation matrix, passed as an array.

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set Visibility

This function changes the visibility of the segment named in the input parameter. If VISIBILITY is GKS$K_VISIBLE (1) and dynamic modification flag DMAF_VISIBILITY is GKS$K_IMM (1), or if VISIBILITY is GKS$K_INVISIBLE (0) and DMAF_INVISIBILITY is GKS$K_IMM (1), then the segment is made visible or invisible immediately, and primitives outside of segments remain on the display surface. Otherwise the following occurs:

- If the REGEN_MODE is GKS$K_IRG_ALLOWED (1), then the image is regenerated. Primitives outside of segments are no longer shown on the display surface.
- If the REGEN_MODE is GKS$K_IRG_SUPPRESSED (0), the regeneration is suppressed and NEW_FRAME is set to GKS$K_NEWFRAME_NECESSARY (1).

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.
SEGMENT_NAME	Integer. The name of the segment whose visibility is changed.
VISIBILITY	Integer. The visibility. Either: • GKS$K_INVISIBLE (0) • GKS$K_VISIBLE (1)

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set Segment Priority

This function assigns the specified priority to the segment. The segment priority is only meaningful if your device supports more than one segment priority.

Segment priority is used when you display overlapping segments. There, the higher-priority segment overlaps the others. The priority is also used to determine which of two or more overlapping segments are selected in pick input. If the overlapping segments all share the same priority, then the pick result is implementation dependent.

If the dynamic modification flag DMAF_SEGMENT_PRIORITY is GKS$K_IMM (1), then the display is changed to reflect changes in overlapping due to the change in priorities immediately, and primitives outside of segments remain on the display surface. Otherwise the following occurs:

- If the REGEN_MODE is GKS$K_IRG_ALLOWED (1), then the image is regenerated. Primitives outside of segments are no longer shown on the display surface.
- If the REGEN_MODE is GKS$K_IRG_SUPPRESSED (0), the regeneration is suppressed and NEW_FRAME is set to GKS$KNEWFRAME_NECESSARY (1).

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.
SEGMENT_NAME	Integer. The name of the segment whose priority is changed.
PRIORITY	Real. The new priority.

Set Segment Priority

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set Detectability

This function assigns the detectability to the segment. Detectability is used in pick input. You can only pick segments with the detectability setting GKS$K_DETECTABLE (1) and the visibility setting GKS$K_VISIBLE (1).

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.
SEGMENT_NAME	Integer. The name of the segment whose detectability will be changed.
DETECTABILITY	Integer. The new detectability. Either: • GKS$K_UNDETECTABLE (0) • GKS$K_DETECTABLE (1)

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Set Highlighting

This function assigns the specified highlighting value to the segment. If a segment is visible and the HIGHLIGHTING field is set to GKS$K_SEGMENT_HIGHLIGHTED (1), the primitives in the segment are able to be highlighted in an implementation-dependent fashion. If the segment is invisible or HIGHLIGHTING is set to GKS$K_SEGMENT_NORMAL, no highlighting occurs.

If the dynamic modification flag DMAF_HIGHLIGHTING is GKS$K_IMM (1), then the segment is highlighted immediately, and primitives outside of segments remain on the display surface. Otherwise the following occurs:

- If the REGEN_MODE is GKS$K_IRG_ALLOWED (1), then the image is regenerated. Primitives outside of segments are no longer shown on the display surface.
- If the REGEN_MODE is GKS$K_IRG_SUPPRESSED (0), the regeneration is suppressed and NEW_FRAME is set to GKS$K_NEWFRAME_NECESSARY (1).

Required

Required for OUTPUT, OUTIN, and MO workstations where the workstation handler supports segments.

Input Parameters

WSL	The address of the handler's local data area.
SEGMENT_NAME	Integer. The name of the segment where highlighting should be enabled or disabled.
HIGHLIGHTING	Integer. The new highlighing mode. Either: • GKS$K_SEGMENT_NORMAL (0) • GKS$K_SEGMENT_HIGHLIGHTED (1)

Output Parameters

None.

Status Codes

Code	Meaning
GKS$_SUCCESS	Success.

Appendix A
Transformations

Transformations convert the data points from WC space to LDC space. Transformations are performed during operations that draw on the display surface or return the coordinates of points on the display surface. This appendix discusses the process of performing transformations in the following two cases:

- When your handler supports segments directly.
- When the GKS kernel simulates segments.

A.1 Concatenating Transformation Matrixes

The kernel passes points in WC when it calls a function that requires the handler to generate output. The handler generates points in NDC during input operations that return coordinates.

The GKS standard defines up to five transformations which must be done on a set of data points to convert them between WC and LDC. Each transformation is expressed as a matrix multiplication. To go from WC to LDC, each point must be transformed using all five transformations. This would require five matrix multiplications on each point, and that process would be very slow.

A more efficient technique is to compose the five transformation matrixes into a single matrix. In this case, instead of performing five matrix multiplications on each point, the handler performs the matrix multiplications once, then uses the results to perform a single matrix multiplication on every data point.

The kernel passes the transformation matrixes to your handler, either implicitly or explicitly, in the following functions:

- Set Normalization Transformation—Defines the WC-NDC transformation.
- Set Workstation Window, Set Workstation Viewport—Define the NDC-LDC transformation.
- Set Segment Transformation—Defines the segment transformation. Used only if the handler supports segments.
- Set NDC Transformation—Establishes additional transformation for segment-related functions.

The handler uses this information to compose the transformations in the most efficient manner.

A.2 NDC Transformation and Segment Simulation

The NDC transformation function is unique to the workstation handler system. It occurs in the transformation operation in two different places, depending on whether the handler supports segments, or the kernel simulates segments.

When the kernel simulates segments for your handler, it stores a segment transformation for each segment. If the kernel determines that the handler needs information about the segment transformation, it passes the information to the handler in the NDC transformation. When a segment is displayed, the handler must apply the NDC transformation to the segment after the WC-NDC transformation and before the NDC-LDC transformation.

The NDC transformation is also used during the Insert Segment function. When the GKS kernel performs an Insert Segment function, the NDC transformation includes information about the open segment transformation, the insert segment transformation, and the segment transformation of the inserted segment.

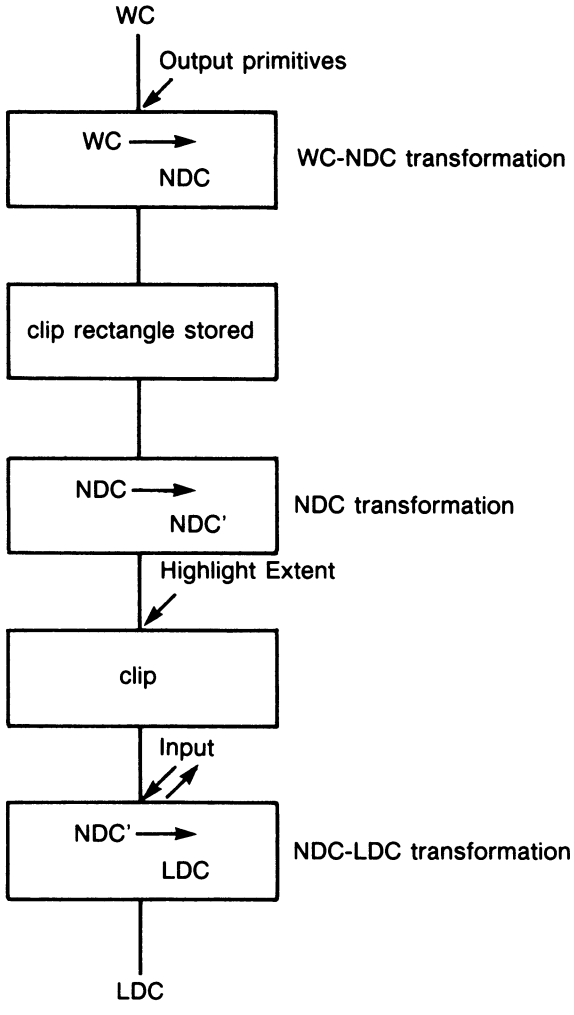
In either of these cases, the NDC transformation is a matrix multiplication, applied as follows:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

In this formula, x and y are the points in NDC, matrix M is the NDC transformation, and x′ and y′ are the points in NDC after the transformation.

The order of transformation operations is called a pipeline. The pipeline for handlers that use segment simulation is shown in Figure A-1.

Figure A–1: Transformation Pipeline for Segment Simulation



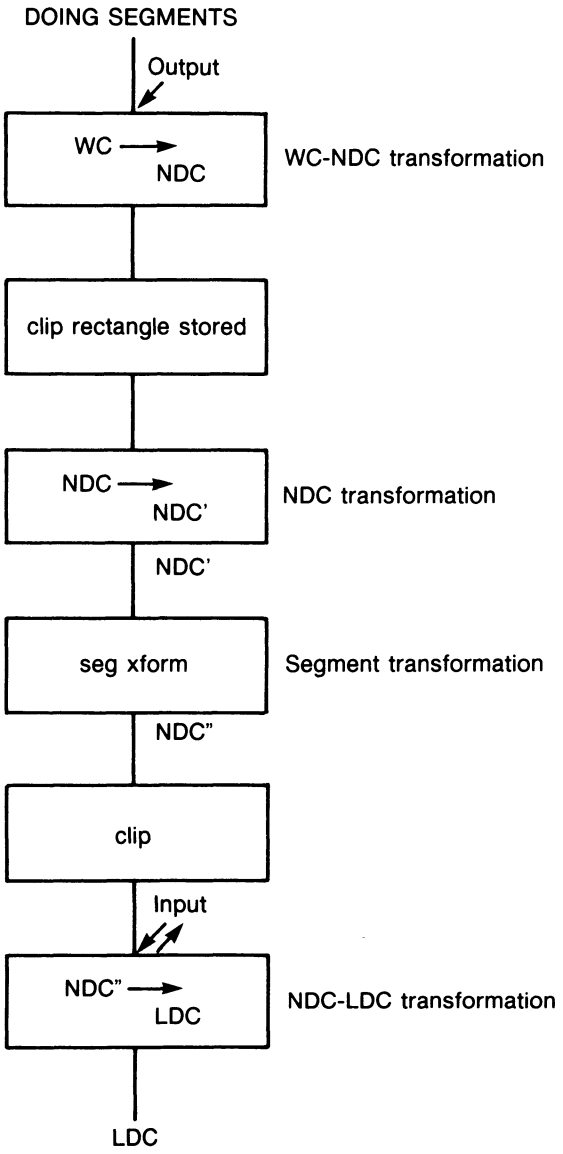
A.3 NDC Transformation When Your Handler Supports Segments

When the device handler supports segments, the kernel passes it the Open Segment transformation in a call to the Set Segment Transformation function. If there has been no call to this function, the handler should use the default segment transformation.

Since the handler already has the segment transformation, the NDC transformation is only used during the Insert Segment function. In this case, the NDC transformation will contain information about the Insert Segment transformation and the segment transformation of the inserted segment. In other cases, the NDC transformation will be the unity transformation.

The matrix multiplication is performed as explained in Section A.2, with the exception that matrix M is the concatenation of the segment transformation with the NDC transformation. Figure A-2 shows the transformation pipeline for handlers that support segments.

Figure A–2: Transformation Pipeline for Handlers that Support Segments



A.4 Algorithms for Transformations

Since there are a number of transformation equations, they are presented here already derived. Note that these equations are for the least capable device. Some devices may be capable of performing all or some of these equations themselves.

A.4.1 Transformations Assuming an Identity NDC Transformation

The NDC transformation is the identity transformation in many cases. If the NDC transformation is the identity, the transformations can be simplified a great deal. Therefore, your handler should test for the unity transformation.

The following is the derivation of the valid transformations assuming the identity NDC transformation. These allow GKS conversions from WC to NDC, from WC to LDC, and from NDC to LDC. The notation ndc refers to the label NDC' in Figure A-1, because it is the identity transformation, ndc = ndc'.

The variables xmin, ymin, xmax, and ymax are indexes into arrays holding the various boundaries.

The variable names are defined as follows:

world_xratio	Ratio of the world viewport to the world window
world_xoffset	Offset of the world window to the world viewport
world_view	Values of the world viewport
world_window	Values of the world window
ws_xratio	Ratio of the workstation viewport to the workstation window
ws_xoffset	Offset of the workstation viewport to the workstation window
ws_window	Values of the workstation window
ws_view	Values of the workstation viewport
xratio	Composite of world_xratio with ws_xratio
offset	Composite of world_xoffset with ws_xoffset
ndc_x, ndc_y	Coordinates of a point in NDC space
ldc_x, ldc_y	Coordinates of a point in LDC space
wc_x, wc_y	Coordinates of a point in WC space

Use the formulas to determine the values of the variables. The equations are derived for the X values only. The equations for the Y values are identical except for the substitution of the Y points.

```
world_xratio = (world_view[xmax] - world_view[xmin])
              /(world_window[xmax] - world_window[xmin]);
world_xoffset = world_window[xmin] - (world_view[xmin] *
                (world_window[xmax] - world_window[xmin])
                /(world_view[xmax] - world_view[xmin]))

ws_xratio = (ws_view[xmax] - ws_view[xmin])/(ws_window[xmax] - ws_window[xmin])

ws_xoffset = ws_view[xmin] - (ws_window[xmin] * ws_xratio)

xratio = world_ratio * ws_xratio

xoffset = world_xoffset * xratio - ws_xoffset
```

Use the formulas to determine the value of a point in the new coordinate system:

```
wc -> ndc   ==>  ndc_x = (wc_x - world_xoffset) * world_xratio;

ndc -> ldc  ==>  ldc_x = (ndc_x * ws_xratio) + ws_xoffset;

wc ->  ldc  ==>  ldc_x = (wc_x * xratio) - xoffset;

ldc ->  wc  ==>  wc_x = (ldc_x + xoffset)/xratio
```

These expressions find the displacement of a point from the origin of the coordinate system (that is, the distance from the 0,0 point), then express this distance as a fraction of the length of a line drawn from the origin, through the point, and continuing to the far edge of the viewport. Then the expressions add the offset of the viewport's origin to yield the point's location on the screen.

Use the formulas for clipping.

xclip_max	The maximum x value of the clipping rectangle
xclip_min	The minimum x value of the clipping rectangle

```
If clipping  is on then the boundary is:
   xclip_min = MAX( ws_window[xmin],world_view[xmin] )
   xclip_max = MIN( ws_window[xmax],world_view[xmax]] )

else if clipping is off then:
   xclip_min = ws_window[xmin]
   xclip_max = ws_window[xmax]
```

For clipping, the boundary may be kept in NDC or DC.

A.4.2 Transformations Assuming the Nonidentity NDC Transformation

If the NDC transformation is not unity, then the handler must incorporate it in the transformation pipelines. The following formulas allow transformations including the NDC transformation. The symbols NDC′ and NDC″ refer to Figure A-2.

A.4.2.1 Transforming from NDC to LDC

The NDC to NDC′ transformation is a simple matrix multiplication done as follows. If the kernel is simulating segments, the matrix M is the NDC transformation. If the handler is managing segments, the matrix M is the composition of the NDC transformation and the segment transformation.

$$\begin{bmatrix} ndc\_x' \\ ndc\_y' \end{bmatrix} == \begin{bmatrix} M1\_1 & M1\_2 & M1\_3 \\ M2\_1 & M2\_2 & M2\_3 \end{bmatrix} * \begin{bmatrix} ndc\_x \\ ndc\_y \\ 1 \end{bmatrix}$$

This becomes the following two equations as defined by matrix multiplication:

```
ndc_x' = M1_1 * ndc_x + M1_2 * ndc_y + M1_3

ndc_y' = M2_1 * ndc_x + M2_2 * ndc_y + M2_3
```

To get the NDC –> LDC transformation, go from NDC to NDC′ to LDC. For the X component, this yields the following:

```
(1) ndx_x -> ndc_x' ==> ndc_x' = M1_1 * ndc_x +
                        M1_2 * ndc_y + M1_3

(2) ndc' -> ldc  ==>  ldc_x' = (ndc_x' * ws_xratio) +
                      ws_xoffset
```

Substituting ndc_x′ in equation 1 for ndc_x′ in equation 2 yields:

```
ldc_x = ( (M1_1 * ndc_x + M1_2 * ndc_y + M1_3) *
          ws_xratio) + ws_xoffset
```

Expanding the equation yields the following:

```
ldc_x = M1_1 * ws_xratio * ndc_x + M1_2 * ws_xratio *
        ndc_y + M1_3 * ws_xratio + ws_xoffset
```

Combine the constants together as follows:

```
xform_xoffset = M1_3 * ws_xratio + ws_xoffset;
xform_x_2 = M1_2 * ws_ratio
xform_x_1 = M1_1 * ws_ratio
```

So the point in ldc is:

```
ldc_x = xform_x_1 * ndc_x + xform_x_2 * ndc_y +
        xform_xoffset
```

As a check, if ndc-> ndc' xform is identity:

```
ldc_x = xform_x_1 * ndc_x + xform_xoffset
       (where xform_x_1 = ws_ratio)
```

This was derived in Section A.4.1.

A.4.2.2 Transforming from WC to LDC

Section A.4.1 showed that the following:

```
(3) wc  -> ndc ==>  ndc_x = (wc_x - world_xoffset) *
                    world_xratio;
```

and the Section A.4.2.1 showed the following:

```
(4) ndc -> dc ==>  dc_x = xform_x_1 * ndc_x +
                   xform_x_2 * ndc_y + xform_xoffset
```

where

```
xform_xoffset = M1_3 * ws_xratio + ws_xoffset;
xform_x_2 = M1_2 * ws_xratio
xform_x_1 = M1_1 * ws_xratio
```

Substituting equation (3) for ndc_x in equation (4) yields the following:

```
(5) ldc_x = xform_x_1 * ((wc_x - world_xoffset) *  world_xratio)
          + xform_x_2 * ndc_y + xform_xoffset
```

By equation 3,

```
ndc_y = (wc_y - world_yoffset) * world_yratio
```

Expanding equation 5 and substituting for ndc_y yields the following:

```
ldc_x = ((wc_x - world_xoffset) * world_xratio * xform_x_1) +
        xform_x_2 * world_yratio *
        (wc_y - world_yoffset) + xform_xoffset
```

Combining the constants together yields the following:

```
wc_xform_x_1 = xform_x_1 * world_xratio
wc_xform_x_2 = xform_x_2 * world_xratio
```

Substituting the new constants yields the following:

```
ldc_x = ((wc_x - world_xoffset) * wc_xform_x_1) +
        wc_xform_x_2 * (wc_y - world_yoffset) + xform_xoffset
```

Expanding the equation yields the following:

```
ldc_x = (wc_x * wc_xform_x_1 - world_xoffset * wc_xform_x_1) +
        wc_xform_x_2 * wc_y - wc_xform_x_2 * world_yoffset) +
        xform_xoffset
```

Rearranging the terms yields the following:

```
(6) ldc_x = wc_x * wc_form_x_1 + wc_xform_x_2 * wc_y -
            world_xoffset * wc_xform_x_1 - wc_xform_x_2 *
            world_yoffset + xform_xoffset
```

Combining the new constants together yields the following:

```
    wc_xform_xoffset = xform_xoffset - world_xoffset *
                       wc_form_x_1 - wc_xform_x_2 * world_yoffset
```

Substituting the new constants into equation 6 yields the transformation from WC to LDC as follows:

```
ldc_x = wc_x * wc_form_x_1 + wc_form_x_2 *
        wc_y + wc_xform_xoffset
```

As a check, assume the following:

```
  wc_xform_x_1     = ws_xratio * world_xratio
  wc_xform_xoffset = ws_xoffset - (world_xoffset * ws_xratio * world_xratio)
```

If ndc-> ndc' xform is identity, then

```
ldc_x = wc_x * wc_xform_x_1 + wc_xform_xoffset
```

Appendix B

Stroke Text Simulation Routines

DEC GKS provides routines to simulate stroke-precision text for workstation handlers. These routines let you implement stroke precision text using DIGITAL-supplied fonts by writing routines that unbundle the text attributes and call the simulation routines contained in the kernel. The text attributes must be unbundled because bundled values are usually stored in the workstation state list (WSL), and the simulation routines have no access to the WSL.

The stroke-precision text simulation routines are described in this appendix. They are the only supported mechanism for implementing the DIGITAL-supplied stroke fonts. However, you may choose to develop your own fonts and implement direct support for stroke text in your handler.

The text simulation routines require the address of your polyline and fill area routines. This could be a problem for handlers written in Pascal, since Pascal passes functions by a special descriptor called a Bound Procedure Value. Pascal can simply the address (called passing by immediate value), but if you chose this method, the routine must have the Pascal Attribute UNBOUND. This means that the function can only address variables that are contained in the local data structure, or variables that are declared local to the routine. Therefore, if you write your handler in Pascal, you must declare any function which will be passed to the kernel as unbound, and pass it by immediate value.

GKS$SIM_STROKE_TEXT

This routine positions and draws the stroke text specified in the call. It performs all calculations needed for the text.

Syntax

```
GKS$SIM_STROKE_TEXT ( loc_data_ptr,
                      text_pos_x,
                      text_pos_y,
                      character_string,
                      attribute_array,
                      polyline_addr,
                      fill_area_addr)
```

RETURNS: **longword condition value**

Arguments

loc_data_ptr

data type:	**longword integer (signed)**
access:	**read-only**
mechanism:	**by reference**

This is the same local data pointer that the kernel passed to the handler. It is not modified by this routine, but the handler's polyline or fill area routines may modify it when GKS$SIM_STROKE_TEXT calls them.

text_pos_x

text_pos_y

data type: **F-floating**
access: **read-only**
mechanism: **by reference**

The X and Y values of the text starting position in World Coordinates (WC).

character_string

data type: **character string**
access: **read-only**
mechanism: **class S descriptor**

The actual text character string to be written.

attribute_array

data type: **array or record**
access: **read-only**
mechanism: **by reference**

An array or record containing the following items. Note that these items must be the unbundled values. That is, the handler must determine the bundled values by looking into the bundle tables, then pass GKS$SIM_STROKE_TEXT the actual values that the simulation routine should use.

Item	Description
TEXT_INDEX	An integer holding the index into the text bundle table.
TEXT_FONT	The number of a DIGITAL-supported stroke font, integer.
TEXT_PRECISION	The precision of the text to be drawn, integer.
CHARACTER_EXPANSION_FACTOR	The character expansion factor, real.
CHARACTER_SPACING	The character spacing, real.
TEXT_COLOR_INDEX	The index into the color table, integer.
CHARACTER_HEIGHT_X	X component of the character height vector, WC, real.
CHARACTER_HEIGHT_Y	Y component of the character height vector, WC, real.
CHARACTER_WIDTH_X	X component of the character width vector, WC, real.
CHARACTER_WIDTH_Y	Y component of the character width vector, WC, real.
TEXT_PATH	The direction of the string, integer.
TEXT_ALIGNMENT_HORZ	The horizontal text alignment, integer.
TEXT_ALIGNMENT_VERT	The vertical text alignment, integer.
PICK_ID	The active pick id for this primitive, integer.

polyline_addr

data type: **longword integer (signed)**

access: **read-only**

mechanism: **by value**

The address of the handler's polyline function, passed by value (that is, the function address, passed by value, not by Bound Procedure Value).

fill_area_addr

data type: **longword integer (signed)**
access: **read-only**
mechanism: **by value**

The address of the handler's fill area function, passed by value (that is, the function address passed by value, not by Bound Procedure Value).

Error Messages

Status Code	Message
GKS$_ERROR_76	Requested text font is not supported for the specified precision on this workstation.
GKS$_ERROR_101	Invalid character code in string.
DECGKS$_ERROR_NEG_32	Illegal font specification in logical.

GKS$SIM_STROKE_TEXT_EXTENT

This routine performs all calculations needed to describe the text extent and concatenation point of the text given the parameters passed to it.

Syntax

```
GKS$SIM_STROKE_TEXT_EXTENT ( text_pos_x,
                             text_pos_y,
                             character_string,
                             attribute_array,
                             concat_x,
                             concat_y,
                             extent_array_x,
                             extent_array_y)
```

RETURNS: **longword condition value**

Arguments

text_pos_x

text_pos_y

data type:	**F-floating**
access:	**read-only**
mechanism:	**by reference**

The X and Y values of the text starting position in World Coordinates (WC).

character_string

data type: **character string**
access: **read-only**
mechanism: **class S descriptor**

attribute_array

data type: **array or record**
access: **read-only**
mechanism: **by reference**

An array or record containing the following items. Note that these items must be the unbundled values. That is, the handler must determine the bundled values by looking into the bundle tables, then pass GKS$SIM_STROKE_TEXT the actual values that the simulation routine should use.

Item	Description
TEXT INDEX	The index into the text bundle table, integer.
TEXT FONT	The number of a DIGITAL-supported stroke font, integer.
TEXT PRECISION	The precision of the text to be drawn, integer.
CHARACTER_ EXPANSION_FACTOR	The text character expansion factor, real.
CHARACTER SPACING	The text character spacing value, real.
TEXT COLOR INDEX	The index into the color table, integer.
CHARACTER HEIGHT X	X component of the character height vector, WC, real.
CHARACTER HEIGHT Y	Y component of the character height vector, WC, real.
CHARACTER WIDTH X	X component of the character width vector, WC, real.
CHARACTER WIDTH Y	Y component of the character width vector, WC, real.
TEXT_PATH	The direction of the string, integer.
TEXT_ALIGNMENT_ HORZ	The horizontal text alignment, integer.
TEXT_ALIGNMENT_ VERT	The vertical text alignment, integer.
PICK_ID	The active pick id for this primitive, integer.

concat_x

concat_y

data type: **F-floating**
access: **write-only**
mechanism: **by reference**

The X and Y value of the concatenation point in WC.

extent_array_x

extent_array_y

data type: **array of four reals**
access: **write-only**
mechanism: **by reference**

Arrays of X and Y values of the four corners of the text extent array in WC, starting with the corner with the lowest X and Y coordinates, and moving counterclockwise around the rectangle.

Error Messages

Status Code	Message
GKS$_ERROR_76	Requested text font is not supported for the specified precision on this workstation.
GKS$_ERROR_101	Invalid character code in string.
DECGKS$_ERROR_NEG_32	Illegal font specification in logical.

GKS$SIM_STROKE_INQ_TEXT_FAC

This routine lets the handler inquire what font numbers are available using the simulation routines, so that it can respond accurately to inquire text facilities calls.

Syntax

```
GKS$SIM_STROKE_INQ_TEXT_FAC ( array_length,
                              font_numbers_array,
                              font_count_returned,
                              font_count)
```

RETURNS: **longword condition value**

Arguments

array_length

data type:	**longword integer (signed)**
access:	**read-only**
mechanism:	**by reference**

The length of the FONT_NUMBERS_ARRAY.

font_number_array

data type:	**longword integer (signed)**
access:	**read/write**
mechanism:	**by reference**

An array of integers of sufficient length to hold the font numbers of the available fonts. Should be 30 to 50 elements long, and is allocated and passed by the handler.

font_count_returned

data type: **longword integer (signed)**

access: **write-only**

mechanism: **by reference**

The number of fonts returned. The returned value is less than or equal to the array length.

font_count

data type: **longword integer (signed)**

access: **write-only**

mechanism: **by reference**

The actual number of fonts that were found. The returned value may be more than FONT_COUNT_RETURNED.

Error Messages

Status Code	Message
GKS$SUCCESS	Success.
DECGKS$_ERROR_NEG_32	Illegal font specification in logical.

Appendix C

Pick Simulation Functions

These functions simulate segment operations for pick input routines. You should use them to implement pick input if your handler does not support segments directly.

The function GKS$FIND_SEGMENT is useful in pick input operations. With this function, you can use your LOCATOR input algorithm to get pick input.

The function GKS$FIND_SEG_EXTENT accepts a segment name and pick id, and returns the segment's boundaries. You should use it as part of your Initialize Pick Input function to determine the initial cursor position.

GKS$FIND_SEGMENT

This function finds the highest priority segment within a rectangle defined by the input parameters, and returns the segment's name and extent rectangle. Your functionpasses a point and a distance from that point, then GKS$FIND_SEGMENT returns the name of the highest priority segment within the rectangle. It also returns the pick id, the segment boundaries, and the primitive extent. If it finds no segment in the area defined by the point and the distance, it returns the value FALSE. Otherwise it returns the value TRUE.

The input rectangle is defined by the parameters X, Y, APERTURE_X, and APERTURE_Y. X and Y define the center point of the rectangle. Its top and bottom edges are distance APERTURE_Y from the input point, in both the up and down directions, and its sides are distance APERTURE_X from the point in both the right and left directions.

The X and Y coordinates are in NDC, after the segment transformation. The aperture coordinates are in NDC with no segment transformation applied to them.

Syntax

```
GKS$FIND_SEGMENT ( loc_data_ptr,
                   x,
                   y,
                   aperture_x,
                   aperture_y,
                   seg_name,
                   pick_id,
                   seg_extent,
                   pick_extent,
                   prim_extent)

RETURNS: GKS$K_TRUE (1) if a segment is found
         GKS$K_FALSE (0) if no segment is found
```

Arguments

loc_data_ptr

data type: **longword integer (signed)**
access: **read-only**
mechanism: **by reference**

The local storage area assigned to the workstation containing the segment.

x

y

data type: **F-floating**
access: **read-only**
mechanism: **by reference**

The X and Y coordinates of the input point in NDC.

aperture_x

aperture_y

data type: **F-floating**
access: **read-only**
mechanism: **by reference**

The dimension of the aperture, in the X and Y dimension. The aperture is a rectangle whose top and bottom are distance APERTURE_Y from the input point, in both the up and down directions, and whose sides are distance APERTURE_X from the point in both the right and left directions. The points are expressed in NDC.

seg_name

data type: **longword integer (signed)**
access: **write-only**
mechanism: **by reference**

The name of the segment found by the function, or zero if no segment was found.

pick_id

data type: **longword integer (signed)**
access: **write-only**
mechanism: **by reference**

The pick id of the segment found by the function, or zero if no segment was found.

seg_extent

data type: **array of eight F-floating**
access: **write-only**
mechanism: **by reference**

The segment extent rectangle in NDC, ordered as (X1,Y1), (X2,Y2), (X3,Y3), (X4,Y4).

pick_extent

data type: **array of eight F-floating**
access: **write-only**
mechanism: **by reference**

The pick extent rectangle in NDC, ordered as (X1,Y1), (X2,Y2), (X3,Y3), (X4,Y4).

prim_extent

data type:	**array of eight F-floating**
access:	**write-only**
mechanism:	**by reference**

The primitive extent rectangle in NDC, ordered as (X1,Y1), (X2,Y2), (X3,Y3), (X4,Y4).

GKS$FIND_SEG_EXTENT

This function accepts a segment name and pick id, and returns the segment extent rectangle and pick extent rectangle. Both rectangles are passed as NDC points after the segment transformation. Your Initialize Pick Input function can use this function to find the initial segment. It should pass a segment name and pick id, then place the cursor within the borders that this function returns.

Syntax

```
GKS$FIND_SEG_EXTENT ( loc_data_ptr,
                      segment_name,
                      pick_id,
                      seg_extent,
                      pick_extent)
```

RETURNS: **GKS$K_TRUE (1) the segment is found**
GKS$K_FALSE (0) if the segment or pick id is not found

Arguments

loc_data_ptr

data type:	**longword integer (signed)**
access:	**read-only**
mechanism:	**by reference**

The local storage area assigned to the workstation containing the segment.

seg_name

data type:	**longword integer (signed)**
access:	**read-only**
mechanism:	**by reference**

The name of the segment whose extent should be found.

pick_id

data type: **longword integer (signed)**
access: **read-only**
mechanism: **by reference**

The pick id of the segment whose extent should be found.

seg_extent

data type: **array of eight F-floating**
access: **write-only**
mechanism: **by reference**

The segment extent rectangle in NDC, ordered as (X1,Y1), (X2,Y2), (X3,Y3), (X4,Y4).

pick_extent

data type: **array of eight F-floating**
access: **write-only**
mechanism: **by reference**

The pick extent rectangle in NDC, ordered as (X1,Y1), (X2,Y2), (X3,Y3), (X4,Y4).

Appendix D

Workstation Handler Function Examples

This appendix contains programming examples for workstation handler systems. You can pattern your functions and data structures on these samples. Note that your actual code will be different so you can take advantage of your device's functionality.

D.1 Data Structures

```
C     This module contains the structure definitions for the
C     Workstation Description Table ( WDT )

C     Predefined polyline bundle structure
      STRUCTURE /predef_pline_bundles/
          INTEGER*4 line_type
          REAL*4    linewidth_scale_factor
          INTEGER*4 color_index
        END STRUCTURE

C     Predefined polymarker bundle structure
      STRUCTURE /predef_pmark_bundles/
          INTEGER*4 marker_type
          REAL*4    msize_scale_factor
          INTEGER*4 color_index
      END STRUCTURE

C     Predefined text bundle structure
      STRUCTURE /predef_text_bundles/
          STRUCTURE /font_prec/ list_font_prec
           INTEGER*4 font
           INTEGER*4 prec
          END STRUCTURE
          REAL*4    char_exp_factor
          REAL*4    char_space
          INTEGER*4 color_index
      END STRUCTURE
```

```
C     Predefined fill bundle structure
      STRUCTURE /predef_fill_bundles/
          INTEGER*4 fill_intstyle
          INTEGER*4 fill_style_ind
          INTEGER*4 color_index
      END STRUCTURE
C     Predefined pattern bundle structure
      STRUCTURE /predef_patt_reps/
          INTEGER*4 patt_dim_x
          INTEGER*4 patt_dim_y
          INTEGER*4 patt_array(2,2)
      END STRUCTURE

C     Predefined color bundle structure
      STRUCTURE /predef_color_reps/
          REAL*4    red
          REAL*4    green
          REAL*4    blue
      END STRUCTURE

C     List of Generalized Drawing Primitives ( GDP ) structure
      STRUCTURE /list_gdp/
          INTEGER*4 gdp_id
          INTEGER*4 gdp_attributes
      END STRUCTURE

C     The Workstation Description Table ( WDT )

      STRUCTURE /wsdt_struct/

C           Workstation type is VT125 (Black and White)
          INTEGER*4 workstation_type /GKS$K_VT125BW/

C           Workstation category is OUTPUT only
          INTEGER*4 workstation_category /GKS$K_WSCAT_OUTPUT/
C           Device coordinate units is OTHER
            INTEGER*4 dev_coordinate_units /GKS$K_OTHER_UNITS/

C           Display space size in device coordinate units
            REAL*4    dev_display_space_size_x  / 767.0 /
            REAL*4    dev_display_space_size_y  / 479.0/

C           Display space size in raster units
            INTEGER*4 raster_display_space_size_x / 768 /
            INTEGER*4 raster_display_space_size_y / 480 /

C           Type of device is RASTER
            INTEGER*4 display_type /GKS$K_WSCLASS_RASTER/
C           Dynamic modification for polyline bundle representation is IRG
            INTEGER*4 dmaf_polyline /GKS$K_IRG/

C           Dynamic modification for polymarker bundle representation is IRG
          INTEGER*4 dmaf_polymarker /GKS$K_IRG/

C           Dynamic modification for Text bundle representation is IRG
          INTEGER*4 dmaf_text /GKS$K_IRG/

C           Dynamic modification for fill bundle representation is IRG
          INTEGER*4 dmaf_fill /GKS$K_IRG/

C           Dynamic modification for pattern bundle representation is IRG
          INTEGER*4 dmaf_pattern /GKS$K_IRG/
```

```
C          Dynamic modification for color bundle representation is IMM
          INTEGER*4 dmaf_color /GKS$K_IMM/
C          Workstation transformation is IRG
           INTEGER*4 dmaf_ws_transformation /GKS$K_IRG/
C          Segment highlighting is IRG
           INTEGER*4 dmaf_highlighting /GKS$K_IRG/
C          Default deferral mode is ASAP (As Soon As Possible)
           INTEGER*4 def_defer_mode /GKS$K_ASAP/

C          Default implicit regeneration mode is SUPPRESSED
           INTEGER*4 regen_mode /GKS$K_IRG_SUPPRESSED/

C          Number of available linetypes is 4
           INTEGER*4 num_linetypes /4/

C          The list of available linetypes is SOLID, DASHED, DOTTED, DASH-DOTTED
           INTEGER*4 list_line_types(4)

C          Number of available linewidths is 1
          INTEGER*4 num_linewidths /1/

C          Nominal linewidth is 1.0
          REAL*4    nominal_linewidth /1.0/

C          Minimum linewidth is 1.0
           REAL*4    minimum_linewidth /1.0/

C          Maximum linewidth is 1.0
           REAL*4    maximum_linewidth /1.0/

C          Number of predefined polyline bundles is 5
           INTEGER*4 number_predef_pline_ind /5/

          RECORD /predef_pline_bundles/ pline_bundles (5)

C          Number of available markertypes is 5
          INTEGER*4 num_markertypes /5/

C          List of available marker types is DOT, PLUS, ASTERISK, CIRCLE,
C          and DIAGONAL CROSS
           INTEGER*4 list_markertypes(5)

C          Number of available marker sizes is 1
          INTEGER*4 num_msizes /1/

C          Nominal marker size is 1.0
          REAL*4    nominal_msize /1.0/

C          Minimum marker size is 1.0
           REAL*4    minimum_msize /1.0/

C          Maximum marker size is 1.0
           REAL*4    maximum_msize /1.0/

C          Number of predefined polymarker bundles is 5
           INTEGER*4 number_predef_pmark_ind /5/

           RECORD /predef_pmark_bundles/ pmark_bundles (5)

C          Number of text font and precision pairs - not used as only
C         text simulation is done.
           INTEGER*4 num_font_prec_pairs  /-1/
```

```
C        List of text font and precision pairs
           INTEGER*4 list_font(50)
           INTEGER*4 list_prec(50)

C          Number of available character expansion factors
           INTEGER*4 num_char_exp_factors /1/
           REAL*4    minimum_char_exp_factor /1.0/
           REAL*4    maximum_char_exp_factor /1.0/

         INTEGER*4 num_char_heights /16/
           REAL*4    minimum_char_height /12.0/
           REAL*4    maximum_char_height /160.0/
           INTEGER*4 num_predef_text_ind /2/

         RECORD /predef_text_bundles/ text_bundles (2)

C          Number of available fill area interior styles is 1
           INTEGER*4 num_fill_intstyle /1/

C          List of fill area interior styles is HOLLOW
           INTEGER*4 list_fill_intstyle /1/

C          Number of available hatch styles is 0
           INTEGER*4 num_hatch_style /0/

C          Number of predefined fill area bundles is 5
           INTEGER*4 num_predef_fill_ind /5/

           INTEGER*4 list_hatch_style(1)
           RECORD /predef_fill_bundles/ fill_bundles (5)

C          Number of predefined pattern indices is 0 - pattern
C         is not supported.
           INTEGER*4 num_predef_patt_ind /0/
           RECORD /predef_patt_reps/ patt_bundles (1)

C          Number of available colors or intensities is 4
         INTEGER*4 num_colors /4/

C          Color available is COLOR
           INTEGER*4 color_available /GKS$K_COLOR/

C          Number of predefined color representations is 4
           INTEGER*4 num_predef_color_rep /4/

              RECORD /predef_color_reps/ color_table (4)

C          Number of available generalized drawing primitives is 0
           INTEGER*4 num_gdp /0/

           RECORD /list_gdp/ list_of_gdp (1)

         INTEGER*4 bundles_initialized / 0 /
         INTEGER*4 max_pline_bundles / 5 /
         INTEGER*4 max_pmark_bundles / 5 /
         INTEGER*4 max_text_bundles / 2 /
         INTEGER*4 max_fill_bundles / 5 /
         INTEGER*4 max_patt_ind / 1 /
         INTEGER*4 max_color_ind / 4 /
     END STRUCTURE
```

```
C     This module contains the Workstation State List ( WSL )
C     The polyline bundle structure
      STRUCTURE /pline_bundles/
          INTEGER*4 pline_index
          INTEGER*4 line_type
          REAL*4    linewidth_scale_factor
          INTEGER*4 color_index
      END STRUCTURE
C     The polymarker bundle structure
      STRUCTURE /pmark_bundles/
          INTEGER*4 pmark_index
          INTEGER*4 marker_type
          REAL*4    msize_scale_factor
          INTEGER*4 color_index
      END STRUCTURE
C     The text bundle structure
      STRUCTURE /text_bundles/
          INTEGER*4 text_index
          STRUCTURE /font_prec_pair/ list_font_prec
           INTEGER*4 font
           INTEGER*4 prec
          END STRUCTURE
          REAL*4    char_exp_factor
          REAL*4    char_space
          INTEGER*4 color_index
      END STRUCTURE

C     The fill bundle structure
      STRUCTURE /fill_bundles/
          INTEGER*4 fill_index
          INTEGER*4 fill_intstyle
          INTEGER*4 fill_style_ind
          INTEGER*4 color_index
      END STRUCTURE

C     The pattern bundle structure
      STRUCTURE /patt_bundles/
          INTEGER*4 patt_index
          INTEGER*4 patt_dim_x
          INTEGER*4 patt_dim_y
          INTEGER*4 patt_array(2,2)
      END STRUCTURE

C     The color bundle structure
      STRUCTURE /color_bundle/
          INTEGER*4 color_index
          REAL*4 red
          REAL*4 green
          REAL*4 blue
      END STRUCTURE
```

```
C     The Workstation State List data structure.
      STRUCTURE /ws_state_list/
          INTEGER*4 begin_structure
          INTEGER*4 unit_num
          INTEGER*4 channel
          CHARACTER*80 connection_id
          INTEGER*4 wstype
          INTEGER*4 segment_set
          INTEGER*4 defer_mode
          INTEGER*4 regen_mode
          INTEGER*4 display_empty
          INTEGER*4 new_frame
          INTEGER*4 global_interactions_present
          INTEGER*4 transform_flag
          REAL*4    ndc_matrix(3,3)
          INTEGER*4 clip_flag
          REAL*4    clip_rectangle(1:4)
          INTEGER*4 number_pline_ind
          RECORD /pline_bundles/ set_pline_bundles (5)
          RECORD /pline_bundles/ real_pline_bundles (5)

          INTEGER*4 number_pmark_ind
          RECORD /pmark_bundles/ set_pmark_bundles (5)
          RECORD /pmark_bundles/ real_pmark_bundles (5)

          INTEGER*4 num_text_ind
          RECORD /text_bundles/ set_text_bundles (2)
          RECORD /text_bundles/ real_text_bundles (2)

          INTEGER*4 num_fill_ind
          RECORD /fill_bundles/ set_fill_bundles (5)
          RECORD /fill_bundles/ real_fill_bundles (5)

          INTEGER*4 num_patt_ind
          RECORD /patt_bundles/ set_patt_bundles (1)
          RECORD /patt_bundles/ real_patt_bundles (1)

          INTEGER*4 num_color_ind
          RECORD /color_bundle/ set_color_table (4)
          RECORD /color_bundle/ real_color_table (4)

          REAL*4 world_window(4)
          REAL*4 world_viewport(4)

          REAL*4 req_ws_window(4)
          REAL*4 req_ws_viewport(4)
          REAL*4 cur_ws_window(4)
          REAL*4 cur_ws_viewport(4)

          REAL*4 trans_matrix(3,3)
          INTEGER*4 end_structure
      END STRUCTURE

C     INDIV_ATTRIBUTES.FOR
C     The escape data record - note none currently supported.
      STRUCTURE /ESCAPE_DATA_RECORD/
      END STRUCTURE
```

```
C     The attribute array and related structures
C     The polyline attributes.
      STRUCTURE /LINE_ATTR/
         INTEGER*4 polyline_index
         INTEGER*4 line_type
         REAL*4    linewidth_scale_factor
         INTEGER*4 polyline_color_index
         INTEGER*4 pick_id
      END STRUCTURE
C     The polyline attributes.
      STRUCTURE /MARKER_ATTR/
         INTEGER*4 polymarker_index
         INTEGER*4 markertype
         REAL*4    markersize_scale_factor
         INTEGER*4 polymarker_color_index
         INTEGER*4 pick_id
      END STRUCTURE

C     The polyline attributes.
      STRUCTURE /FILL_ATTR/
         INTEGER*4 fill_area_index
         INTEGER*4 interior_style
         INTEGER*4 fill_style_index
         INTEGER*4 fill_area_color_index
         REAL*4    pattern_refernce_point_x
         REAL*4    pattern_refernce_point_y
         REAL*4    pattern_height_x
         REAL*4    pattern_height_y
         REAL*4    pattern_width_x
         REAL*4    pattern_width_y
         INTEGER*4 pick_id
      END STRUCTURE

C     The polyline attributes.
      STRUCTURE /TEXT_ATTR/
         INTEGER*4 text_index
         INTEGER*4 font
         INTEGER*4 precision
         REAL*4    char_exp_factor
         REAL*4    char_space
         INTEGER*4 text_color_index
         REAL*4    char_height_x
         REAL*4    char_height_y
         REAL*4    char_width_x
         REAL*4    char_width_y
         INTEGER*4 text_path
         INTEGER*4 text_align_horiz
         INTEGER*4 text_align_vert
         INTEGER*4 pick_id
      END STRUCTURE

C     The attribute structure.
      STRUCTURE /ATTRIBUTES/
         RECORD /LINE_ATTR/ line_attributes
         RECORD /MARKER_ATTR/ marker_attributes
         RECORD /FILL_ATTR/ fill_attributes
         RECORD /TEXT_ATTR/ text_attributes
         INTEGER*4 cell_array_pick_id
      END STRUCTURE
```

D.2 Control Functions

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Open Workstation"
C  Input Parameters:
C     wsl          workstation state list
C     wstype           workstation type
C     devnam           device name
C     ws_id              workstation identifier
C
C  Output Parameters:
C     level        GKS level of handler
C
C  Value Returned:
C     GKS$_SUCCESS       Success
C     GKS$_ERROR_26      Specified workstation cannot be opened
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

         INTEGER*4 FUNCTION HANDLER__OPEN_WS( wsl, wstype, devname,
      +              ws_id, level)

      IMPLICIT NONE
      INCLUDE 'sys$library:gksdefs.for/nolist'
         INCLUDE 'wsdt.for/nolist'
         INCLUDE 'ws_state_list.for/nolist'
      INCLUDE 'indiv_attributes.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'
      INCLUDE 'ssdef.for/nolist'

C      Parameters
         RECORD /ws_state_list/ wsl
         INTEGER*4 wstype
         CHARACTER*(*) devname
      INTEGER*4 ws_id
         INTEGER*4 level
         RECORD /wsdt_struct/ wsdt
         COMMON / workdesc / wsdt
      RECORD / attributes / dummy_att_array
      INTEGER*4 i, j, total_rows,total_columns, status
      INTEGER*4 dummy_var / 0 /

      CHARACTER*3 clear_text
      INTEGER*4 HANDLER__INQ_PREDEF_PLINE_REP
      INTEGER*4 HANDLER__INQ_PREDEF_PMARK_REP
      INTEGER*4 HANDLER__INQ_PREDEF_TEXT_REP
      INTEGER*4 HANDLER__INQ_PREDEF_PATT_REP
      INTEGER*4 HANDLER__INQ_PREDEF_FILL_REP
      INTEGER*4 HANDLER__INQ_PREDEF_COLOR_REP
      INTEGER*4 HANDLER__SET_PLINE_REP
      INTEGER*4 HANDLER__SET_PMARKER_REP
      INTEGER*4 HANDLER__SET_TEXT_REP
      INTEGER*4 HANDLER__SET_PATT_REP
      INTEGER*4 HANDLER__SET_FILL_REP
      INTEGER*4 HANDLER__SET_COLOR_REP
      INTEGER*4 LIB$GET_LUN
      INTEGER*4 SYS$TRNLNM
```

```
C     Wake up the debugger if the debugging logical is defined

      status=SYS$TRNLNM( %VAL(0),'LNM$FILE_DEV',
     +         'GKS$$HANDLER_DEBUG',%VAL(0),%VAL(0))
      IF ( status .EQ. SS$_NORMAL ) THEN
          CALL LIB$SIGNAL( %VAL(SS$_DEBUG) )
      ELSEIF ( status .NE. SS$_NOLOGNAM ) THEN
          HANDLER__OPEN_WS = status
          RETURN
      ENDIF
      clear_text   = CHAR('1B'X) // '[H'

C     * Find an unused unit number

      wsl.unit_num = 0
      status = LIB$GET_LUN(wsl.unit_num)
      IF (status .NE. 1) GOTO 999

C     * Open Device *

      OPEN (wsl.unit_num, FILE = devname, STATUS = 'UNKNOWN',
     +        ERR = 999, IOSTAT = status)

      GOTO 1000
C     Open error or error on LIB$GET_LUN goes here
 999    HANDLER__OPEN_WS = GKS$_ERROR_26
      RETURN

 1000      CONTINUE
C-------------------------------------------------------------------------
C                                                                        |
C       Initialize the bundle table for workstation description table    |
C                                                                        |
C-------------------------------------------------------------------------
      CALL BUNDLE_INIT()
C-------------------------------------------------------------------------
C                                                                        |
C     Initialize workstation state list                                  |
C                                                                        |
C-------------------------------------------------------------------------

        wsl.connection_id = devname
        wsl.wstype = wstype
        wsl.defer_mode = wsdt.def_defer_mode
        wsl.regen_mode = wsdt.regen_mode

        wsl.display_empty = GKS$K_EMPTY
        wsl.new_frame = GKS$K_NEWFRAME_NOTNECESSARY

        wsl.global_interactions_present = 0

        wsl.transform_flag = GKS$K_NOTPENDING
```

```
      wsl.number_pline_ind = wsdt.number_predef_pline_ind
      DO 10 i = 1,wsl.number_pline_ind
            wsl.set_pline_bundles(i).pline_index = i
       status = handler__inq_predef_pline_rep( wstype, i,
     +            wsl.set_pline_bundles(i).line_type,
     +            wsl.set_pline_bundles(i).linewidth_scale_factor,
     +            wsl.set_pline_bundles(i).color_index )
       status = handler__set_pline_rep( wsl, i,
     +            wsl.set_pline_bundles(i).line_type,
     +            wsl.set_pline_bundles(i).linewidth_scale_factor,
     +            wsl.set_pline_bundles(i).color_index )
10    CONTINUE

      wsl.number_pmark_ind = wsdt.number_predef_pmark_ind
      DO 20 i = 1,wsl.number_pmark_ind
           wsl.set_pmark_bundles(i).pmark_index = i
           status = handler__inq_predef_pmark_rep ( wstype, i,
     +             wsl.set_pmark_bundles(i).marker_type,
     +             wsl.set_pmark_bundles(i).msize_scale_factor,
     +             wsl.set_pmark_bundles(i).color_index )
           status = handler__set_pmarker_rep ( wsl, i,
     +             wsl.set_pmark_bundles(i).marker_type,
     +             wsl.set_pmark_bundles(i).msize_scale_factor,
     +             wsl.set_pmark_bundles(i).color_index )
20    CONTINUE

      wsl.num_text_ind = wsdt.num_predef_text_ind
      DO 30 i = 1,wsl.num_text_ind
           wsl.set_text_bundles(i).text_index = i
           status = handler__inq_predef_text_rep( wstype, i,
     +             wsl.set_text_bundles(i).list_font_prec.font,
     +             wsl.set_text_bundles(i).list_font_prec.prec,
     +             wsl.set_text_bundles(i).char_exp_factor,
     +             wsl.set_text_bundles(i).char_space,
     +             wsl.set_text_bundles(i).color_index )
           status = handler__set_text_rep( wsl, i,
     +             wsl.set_text_bundles(i).list_font_prec.font,
     +             wsl.set_text_bundles(i).list_font_prec.prec,
     +             wsl.set_text_bundles(i).char_exp_factor,
     +             wsl.set_text_bundles(i).char_space,
     +             wsl.set_text_bundles(i).color_index )
30    CONTINUE

      wsl.num_fill_ind = wsdt.num_predef_fill_ind
      DO 40 i = 1,wsl.num_fill_ind
          wsl.set_fill_bundles(i).fill_index = i
       status = handler__inq_predef_fill_rep( wstype, i,
     +             wsl.set_fill_bundles(i).fill_intstyle,
     +             wsl.set_fill_bundles(i).fill_style_ind,
     +             wsl.set_fill_bundles(i).color_index)
       status = handler__set_fill_rep( wsl, i,
     +             wsl.set_fill_bundles(i).fill_intstyle,
     +             wsl.set_fill_bundles(i).fill_style_ind,
     +             wsl.set_fill_bundles(i).color_index)
40    CONTINUE
```

```
      wsl.num_patt_ind = wsdt.num_predef_patt_ind
      DO 50 i = 1,wsl.num_patt_ind
           wsl.set_patt_bundles(i).patt_index = i
           status = handler__inq_predef_patt_rep ( wstype, i,
     +             wsl.set_patt_bundles(i).patt_dim_x,
     +             wsl.set_patt_bundles(i).patt_dim_y,
     +             wsl.set_patt_bundles(i).patt_array,
     +             total_rows, total_columns )
           status = handler__set_patt_rep ( wsl, i,
     +             wsl.set_patt_bundles(i).patt_dim_x,
     +             wsl.set_patt_bundles(i).patt_dim_y, 1,
     +             wsl.set_patt_bundles(i).patt_array )
50    CONTINUE

      wsl.num_color_ind = wsdt.num_predef_color_rep
      DO 60 i = 1,wsl.num_color_ind
          wsl.set_color_table(i).color_index = i - 1
       status = handler__inq_predef_color_rep ( wstype, i - 1,
     +             wsl.set_color_table(i).red,
     +             wsl.set_color_table(i).green,
     +             wsl.set_color_table(i).blue)
       status = handler__set_color_rep ( wsl, i - 1,
     +             wsl.set_color_table(i).red,
     +             wsl.set_color_table(i).green,
     +             wsl.set_color_table(i).blue )
60    CONTINUE

C     Initialize the default workstation window and viewport
      wsl.req_ws_window(1) = 0.0
      wsl.req_ws_window(2) = 1.0
      wsl.req_ws_window(3) = 0.0
      wsl.req_ws_window(4) = 1.0

      wsl.cur_ws_window(1) = 0.0
      wsl.cur_ws_window(2) = 1.0
      wsl.cur_ws_window(3) = 0.0
      wsl.cur_ws_window(4) = 1.0

      wsl.req_ws_viewport(1) = 0.0
      wsl.req_ws_viewport(3) = 0.0
      wsl.cur_ws_viewport(1) = 0.0
      wsl.cur_ws_viewport(3) = 0.0
C     Grab biggest square available on the display
     IF ( wsdt.dev_display_space_size_x .gt.
     +          wsdt.dev_display_space_size_y ) THEN
C      x is bigger than y
              wsl.req_ws_viewport(2) = wsdt.dev_display_space_size_y
              wsl.req_ws_viewport(4) = wsdt.dev_display_space_size_y
              wsl.cur_ws_viewport(2) = wsdt.dev_display_space_size_y
              wsl.cur_ws_viewport(4) = wsdt.dev_display_space_size_y
     ELSE
C      y is bigger than x
              wsl.req_ws_viewport(2) = wsdt.dev_display_space_size_x
              wsl.req_ws_viewport(4) = wsdt.dev_display_space_size_x
              wsl.cur_ws_viewport(2) = wsdt.dev_display_space_size_x
              wsl.cur_ws_viewport(4) = wsdt.dev_display_space_size_x
     END IF
```

```
C     Initialize the NDC transformation matrix to the identity
     CALL SET_MATRIX_ID( wsl.ndc_matrix )
C       GKS level is 2c
        level = GKS$K_LEVEL_2C
C     Initialize the graphics on the device

     WRITE(wsl.unit_num,11) clear_text
 11     format('+',A )
     CALL ENTER_GRAPHICS( wsl )
     WRITE(wsl.unit_num,12) 'S[0,0](A[0,0][767,479]I(D)S1T0)S(C0)'
 12     format('+',A )
     WRITE(wsl.unit_num,13) 'W(VI3M1F3N0P1P(M2)S0)'
 13     format('+',A )
         WRITE(wsl.unit_num,14) '@.'
 14     format('+',A )
     WRITE(wsl.unit_num,15) 'T[+9,+0](S1,H2,S[9,20],M[1,2],D0,I0,A0)'
 15     format('+',A )
     WRITE(wsl.unit_num,16) 'S(E)'
 16     format('+',A )

     CALL EXIT_GRAPHICS( wsl )
C       Return status of success.
        HANDLER__OPEN_WS = GKS$_SUCCESS
        RETURN
        END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Bundle_Init"
C
C     This routine initializes the bundle tables in the Workstation
C     Description Table ( WDT ).
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

     SUBROUTINE BUNDLE_INIT()
     IMPLICIT NONE
     INCLUDE 'sys$library:gksdefs.for/nolist'
        INCLUDE 'wsdt.for/nolist'
```

```
      RECORD /wsdt_struct/ wsdt
      COMMON / workdesc / wsdt
    INTEGER*4 i
    INTEGER*4 GKS$SIM_STROKE_INQ_TEXT_FAC
    INTEGER*4 total_fonts
    INTEGER*4 status
C    check if bundles are all already initialized
    if ( wsdt.bundles_initialized .eq.  1 ) then
        return
    end if
C       Bundle table for polyline
       wsdt.pline_bundles(1).line_type = GKS$K_LINETYPE_SOLID
       wsdt.pline_bundles(1).linewidth_scale_factor = 1.0
       wsdt.pline_bundles(1).color_index = 1
       wsdt.pline_bundles(2).line_type = GKS$K_LINETYPE_DASHED
       wsdt.pline_bundles(2).linewidth_scale_factor = 1.0
       wsdt.pline_bundles(2).color_index = 2
       wsdt.pline_bundles(3).line_type = GKS$K_LINETYPE_DASHED
       wsdt.pline_bundles(3).linewidth_scale_factor = 1.0
       wsdt.pline_bundles(3).color_index = 1
       wsdt.pline_bundles(4).line_type = GKS$K_LINETYPE_DOTTED
       wsdt.pline_bundles(4).linewidth_scale_factor = 1.0
       wsdt.pline_bundles(4).color_index = 1
       wsdt.pline_bundles(5).line_type =
     +                        GKS$K_LINETYPE_DASHED_DOTTED
       wsdt.pline_bundles(5).linewidth_scale_factor = 1.0
       wsdt.pline_bundles(5).color_index = 1

       wsdt.list_line_types(1) = GKS$K_LINETYPE_SOLID
       wsdt.list_line_types(2) = GKS$K_LINETYPE_DASHED
       wsdt.list_line_types(3) = GKS$K_LINETYPE_DOTTED
       wsdt.list_line_types(4) = GKS$K_LINETYPE_DASHED_DOTTED

C       Bundle table for polymarkers
       wsdt.pmark_bundles(1).marker_type = GKS$K_MARKERTYPE_DOT
       wsdt.pmark_bundles(1).msize_scale_factor = 1.0
       wsdt.pmark_bundles(1).color_index = 1
       wsdt.pmark_bundles(2).marker_type = GKS$K_MARKERTYPE_PLUS
       wsdt.pmark_bundles(2).msize_scale_factor = 1.0
       wsdt.pmark_bundles(2).color_index = 1
       wsdt.pmark_bundles(3).marker_type = GKS$K_MARKERTYPE_ASTERISK
       wsdt.pmark_bundles(3).msize_scale_factor = 1.0
       wsdt.pmark_bundles(3).color_index = 1
       wsdt.pmark_bundles(4).marker_type = GKS$K_MARKERTYPE_CIRCLE
       wsdt.pmark_bundles(4).msize_scale_factor = 1.0
       wsdt.pmark_bundles(4).color_index = 1
       wsdt.pmark_bundles(5).marker_type =
     +                        GKS$K_MARKERTYPE_DIAGONAL_CROSS
       wsdt.pmark_bundles(5).msize_scale_factor = 1.0
       wsdt.pmark_bundles(5).color_index = 1

    wsdt.list_markertypes(1) =  GKS$K_MARKERTYPE_DOT
    wsdt.list_markertypes(2) =  GKS$K_MARKERTYPE_PLUS
    wsdt.list_markertypes(3) =  GKS$K_MARKERTYPE_ASTERISK
    wsdt.list_markertypes(4) =  GKS$K_MARKERTYPE_CIRCLE
    wsdt.list_markertypes(5) =  GKS$K_MARKERTYPE_DIAGONAL_CROSS
```

```
C       Bundle table for text
      status = GKS$SIM_STROKE_INQ_TEXT_FAC( 50, wsdt.list_font,
     +       wsdt.num_font_prec_pairs, total_fonts )
      DO 10 i = 1,wsdt.num_font_prec_pairs
          wsdt.list_prec(i) = GKS$K_TEXT_PRECISION_STROKE
 10     CONTINUE
        wsdt.text_bundles(1).list_font_prec.font = 1
        wsdt.text_bundles(1).list_font_prec.prec =
     +                     GKS$K_TEXT_PRECISION_STROKE
        wsdt.text_bundles(1).char_exp_factor = 1.0
        wsdt.text_bundles(1).char_space = 0.0
        wsdt.text_bundles(1).color_index = 1
        wsdt.text_bundles(2).list_font_prec.font = 1
        wsdt.text_bundles(2).list_font_prec.prec =
     +                     GKS$K_TEXT_PRECISION_STROKE
        wsdt.text_bundles(2).char_exp_factor = 1.0
        wsdt.text_bundles(2).char_space = 0.0
        wsdt.text_bundles(2).color_index = 2

C       Bundle table for fill area
        wsdt.fill_bundles(1).fill_intstyle = GKS$K_INTSTYLE_HOLLOW
        wsdt.fill_bundles(1).fill_style_ind = 1
        wsdt.fill_bundles(1).color_index = 1
        wsdt.fill_bundles(2).fill_intstyle = GKS$K_INTSTYLE_HOLLOW
        wsdt.fill_bundles(2).fill_style_ind = 1
        wsdt.fill_bundles(2).color_index = 2
        wsdt.fill_bundles(3).fill_intstyle = GKS$K_INTSTYLE_HOLLOW
        wsdt.fill_bundles(3).fill_style_ind = 1
        wsdt.fill_bundles(3).color_index = 3
        wsdt.fill_bundles(4).fill_intstyle = GKS$K_INTSTYLE_HOLLOW
        wsdt.fill_bundles(4).fill_style_ind = 1
        wsdt.fill_bundles(4).color_index = 1
        wsdt.fill_bundles(5).fill_intstyle = GKS$K_INTSTYLE_HOLLOW
        wsdt.fill_bundles(5).fill_style_ind = 1
        wsdt.fill_bundles(5).color_index = 2

C       Bundle table for patterns
        wsdt.patt_bundles(1).patt_dim_x = 0
        wsdt.patt_bundles(1).patt_dim_y = 0
        wsdt.patt_bundles(1).patt_array(1,1) = 1
        wsdt.patt_bundles(1).patt_array(1,2) = 0
        wsdt.patt_bundles(1).patt_array(2,1) = 0
        wsdt.patt_bundles(1).patt_array(2,2) = 1

C       Bundle table for color representation
        wsdt.color_table(1).red = 0.0
        wsdt.color_table(1).green = 0.0
        wsdt.color_table(1).blue = 0.0
        wsdt.color_table(2).red = 0.0
        wsdt.color_table(2).green = 1.0
        wsdt.color_table(2).blue = 0.0
        wsdt.color_table(3).red = 1.0
        wsdt.color_table(3).green = 0.0
        wsdt.color_table(3).blue = 0.0
        wsdt.color_table(4).red = 0.0
        wsdt.color_table(4).green = 0.0
        wsdt.color_table(4).blue = 1.0
```

```
C     bundles are all initialized so set flag
     wsdt.bundles_initialized = 1
     RETURN
     END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Close Workstation"
C  Input Parameters:
C     wsl          address of local data area allocated for driver
C     attrib_array     attribute array with any information needed to close
C                       down the workstation
C
C  Output Parameters:
C     NONE.
C
C  Value Returned:
C     GKS$_SUCCESS     Success
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__CLOSE_WS( wsl, attrib_array)

     IMPLICIT NONE
        INCLUDE 'indiv_attributes.for/nolist'
        INCLUDE 'ws_state_list.for/nolist'
     INCLUDE 'sys$library:gksmsgs.for/nolist'

C     Parameters
        RECORD /ws_state_list/ wsl
        RECORD /attributes/ attrib_array

C       Do any clean-up required - may want to reset terminal attributes

C       Deassign device channel.

     CLOSE ( UNIT = wsl.unit_num )
C       Clear display (if desired)
        HANDLER__CLOSE_WS = GKS$_SUCCESS

        RETURN
        END
```

D.3 Transformation Functions

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Set Normalization Transformation"
C  Input Parameters:
C     wsl          address of local data area allocated for driver
C     window          normalization window limits( XMIN, XMAX, YMIN, YMAX)
C                  in WC
C     viewport           normalization viewport limits( XMIN, XMAX, YMIN,
C                YMAX) in NDC
C     clip_flag          when TRUE implies clipping is enabled, and clipping
C                rectangle is the viewport; when FALSE implies clipping
C                is disabled and the clipping rectangle is [0,1] x
C                [0,1]
C
C  Output Parameters:
C     NONE.
C
C  Value Returned:
C     GKS$_SUCCESS      Success
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__SET_NORM_XFORM( wsl, window,
     +      viewport, clip_flag)

      IMPLICIT NONE
      INCLUDE 'sys$library:gksdefs.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'
        INCLUDE 'ws_state_list.for/nolist'

C      Parameters
        RECORD /ws_state_list/ wsl
        REAL*4 window(1:4)
        REAL*4 viewport(1:4)
        INTEGER*4 clip_flag
      INTEGER*4 i

C       Set normalization window limits.
        DO 10 i = 1,4
            wsl.world_window(i) = window(i)
            wsl.world_viewport(i) = viewport(i)
 10     CONTINUE
```

```
C     Call a routine to update the various transformation variables
C     This uses the formulas specified in Appendix A.
         CALL UPDATE_TRANSFORMATION( wsl )
      wsl.clip_flag = clip_flag
      IF ( clip_flag .NE. GKS$K_NOCLIP ) THEN
         DO 20 i = 1,4
          wsl.clip_rectangle(i) = viewport(i)
 20        CONTINUE
      ELSE
         wsl.clip_rectangle(1) = 0.0
         wsl.clip_rectangle(2) = 1.0
         wsl.clip_rectangle(3) = 0.0
         wsl.clip_rectangle(4) = 1.0
      END IF
C       Recomputing geometric attributes at this point is not
C     necessary as everything is kept in WC and uses simulation

C       Return status of success.
        HANDLER__SET_NORM_XFORM = GKS$_SUCCESS

        RETURN
        END


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Set NDC Transformation"
C  Input Parameters:
C     wsl           address of local data area allocated for driver
C     ndc_transform       NDC transformation array - ordered M(1,1),
C                   M(1,2), M(1,3), M(2,1), M(2,2), M(2,3)
C                   Locations M(1,3) and M(2,3) are in NDC.
C
C  Output Parameters:
C     NONE.
C
C  Value Returned:
C     GKS$_SUCCESS      Success
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__SET_NDC_XFORM( wsl, ndc_transform )

      IMPLICIT NONE
         INCLUDE 'ws_state_list.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'

C      Parameters
         RECORD /ws_state_list/ wsl
         REAL*4 ndc_transform(1:6)
```

```
C     Save the 1 x 6 matrix in a 3 x 3
     wsl.ndc_matrix(1,1) = ndc_transform(1)
     wsl.ndc_matrix(1,2) = ndc_transform(4)
     wsl.ndc_matrix(1,3) = 0.0
     wsl.ndc_matrix(2,1) = ndc_transform(2)
     wsl.ndc_matrix(2,2) = ndc_transform(5)
     wsl.ndc_matrix(2,3) = 0.0
     wsl.ndc_matrix(3,1) = ndc_transform(3)
     wsl.ndc_matrix(3,2) = ndc_transform(6)
     wsl.ndc_matrix(3,3) = 0.0
C     Recomputing geometric attributes at this point is not
C     necessary as everything is kept in WC and uses simulation

C     Call a routine to update the various transformation variables
C     This uses the formulas specified in Appendix A.
        CALL UPDATE_TRANSFORMATION( wsl )
C       Return status of success.
        HANDLER__SET_NDC_XFORM = GKS$_SUCCESS

        RETURN
        END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Set Workstation Window"
C  Input Parameters:
C     wsl           address of local data area allocated for driver
C     window_limits       workstation window limits( XMIN, XMAX, YMIN, YMAX)
C                 in NDC
C
C  Output Parameters:
C     transform_flag     transformation update state either GKS$K_PENDING (0)
C                or GKS$K_NOTPENDING (1)
C
C  Value Returned:
C     GKS$_SUCCESS      Success
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__SET_WS_WINDOW( wsl, window_limits,
     +      transform_flag )

      IMPLICIT NONE
      INCLUDE 'sys$library:gksdefs.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'
      INCLUDE 'ws_state_list.for/nolist'
      INCLUDE 'wsdt.for/nolist'

C      Parameters
        RECORD /ws_state_list/ wsl
        REAL*4 window_limits(1:4)
        INTEGER*4 transform_flag
      INTEGER*4 i
      RECORD / wsdt_struct / wsdt
      COMMON / workdesc / wsdt
C       Set requested workstation window.
        DO 10 i = 1,4
            wsl.req_ws_window(i) = window_limits(i)
10      CONTINUE
```

```
        IF ( ( wsl.display_empty .EQ. GKS$K_EMPTY ) .OR.
     +        ( wsdt.dmaf_ws_transformation .EQ. GKS$K_IMM ) )
     +      THEN
            wsl.transform_flag = GKS$K_NOTPENDING

            DO 20 i = 1,4
                wsl.cur_ws_window(i) = window_limits(i)
 20         CONTINUE

C     Call a routine to update the various transformation variables
C     This uses the formulas specified in Appendix A.
             CALL UPDATE_TRANSFORMATION( wsl )
C         Recomputing geometric attributes at this point is not
C         necessary as everything is kept in WC and uses simulation

C           IF ( wsl.workstation_transformation .EQ. GKS$K_IMM ) THEN
C           do dynamic modification. ( THIS HANDLER CAN'T )

C           Update workstation transformation update state.
            wsl.transform_flag = GKS$K_NOTPENDING
            transform_flag = GKS$K_NOTPENDING
        ELSE IF ( ( wsdt.dmaf_ws_transformation .eq. GKS$K_IRG )
     +       .AND. ( wsl.display_empty .EQ. GKS$K_NOTEMPTY ) ) THEN
             wsl.transform_flag = GKS$K_PENDING
            transform_flag = GKS$K_PENDING
        END IF

C       Return status of success.
        HANDLER__SET_WS_WINDOW = GKS$_SUCCESS

        RETURN
        END


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Set Workstation Viewport"
C  Input Parameters:
C     wsl            address of local data area allocated for driver
C     viewport_limits      workstation viewport limits( XMIN, XMAX, YMIN,
C                       YMAX) in DC
C
C  Output Parameters:
C     transform_flag     transformation update state either GKS$K_PENDING (0)
C               or GKS$K_NOTPENDING (1)
C
C  Value Returned:
C     GKS$_SUCCESS       Success
C     GKS$_ERROR_54      Workstation viewport is not within the display space
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__SET_WS_VIEWPORT( wsl,
     +         viewport_limits, transform_flag )

      IMPLICIT NONE
      INCLUDE 'sys$library:gksdefs.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'
      INCLUDE 'ws_state_list.for/nolist'
      INCLUDE 'wsdt.for/nolist'
```

```
C     Parameters
        RECORD /ws_state_list/ wsl
        REAL*4 viewport_limits(1:4)
        INTEGER*4 transform_flag
      INTEGER*4 i

      RECORD / wsdt_struct / wsdt
      COMMON / workdesc / wsdt
C     Make sure requested viewport is within the display space
      IF ( ( viewport_limits(1) .LT. 0.0 ) .OR.
     +          ( viewport_limits(2) .GT. wsdt.dev_display_space_size_x )
     +          .OR. ( viewport_limits(3) .LT. 0.0 ) .OR.
     +          ( viewport_limits(4) .GT. wsdt.dev_display_space_size_y ) )
     +          THEN
           HANDLER__SET_WS_VIEWPORT = GKS$_ERROR_54
              RETURN
      END IF
C        Set requested workstation viewport.
         DO 10 i = 1,4
             wsl.req_ws_viewport(i) = viewport_limits(i)
10       CONTINUE

C        If dynamic modification accepted for workstation transformation is
C        IMM or if display surface empty is empty:
         IF ( ( wsdt.dmaf_ws_transformation .EQ. GKS$K_IMM ) .OR.
     +        ( wsl.display_empty .EQ. GKS$K_EMPTY )  ) THEN

            DO 20 i = 1,4
                wsl.cur_ws_viewport(i) = viewport_limits(i)
20          CONTINUE

C     Call a routine to update the various transformation variables
C     This uses the formulas specified in Appendix A.
             CALL UPDATE_TRANSFORMATION( wsl )

C          Recomputing geometric attributes at this point is not
C          necessary as everything is kept in WC and uses simulation
C            IF ( wsdt.dmaf_ws_transformation .EQ. GKS$K_IMM ) THEN
C            do dynamic modification. ( THIS HANDLER CAN'T )
             wsl.transform_flag = GKS$K_NOTPENDING
             transform_flag = GKS$K_NOTPENDING
           ELSE IF ( ( wsdt.dmaf_ws_transformation .EQ. GKS$K_IRG )
     +          .AND. ( wsl.display_empty .EQ. GKS$K_NOTEMPTY ) ) THEN
              wsl.transform_flag = GKS$K_PENDING
              transform_flag = GKS$K_PENDING
           END IF

C          Return status of success.
           HANDLER__SET_WS_VIEWPORT = GKS$_SUCCESS

           RETURN
           END
```

D.4 Output Functions

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Polyline"
C  Input Parameters:
C     wsl          address of local data area allocated for driver
C     num_pts          number of points in the line to be drawn ( >= 2)
C     wc_x_array        an array of the x coordinates in WC
C     wc_y_array        an array of the y coordinates in WC
C     temp_x_array      a temporary array of the transformed x coordinates
C     temp_y_array      a temporary array of the transformed y coordinates
C     attrib_array      an array of polyline attributes
C     asf_mask           a 32-bit bitmask holding the attribute source flags
C
C  Output Parameters:
C     NONE.
C
C  Value Returned:
C     GKS$_SUCCESS      Success
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__POLYLINE( wsl, num_pts,
     +     wc_x_array, wc_y_array, temp_x_array, temp_y_array,
     +     attrib_array, asf_mask)

      IMPLICIT NONE
      INCLUDE 'sys$library:gksdefs.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'
         INCLUDE 'ws_state_list.for/nolist'
         INCLUDE 'indiv_attributes.for/nolist'

C     Parameters
        RECORD /ws_state_list/ wsl
        INTEGER*4 num_pts
        REAL*4 wc_x_array(*)
        REAL*4 wc_y_array(*)
        INTEGER*4 temp_x_array(*)
        INTEGER*4 temp_y_array(*)
        RECORD /line_attr/ attrib_array
        INTEGER*4 asf_mask

C---------------------------------------------------------------+
C
C     Declare local variables:
C        index          index into polyline attributes list
C      C$          REGIS command to set color
C      L$          REGIS command to set line type
C      ascii_x     x coordinate in ASCII
C      ascii_y     y coordinate in ASCII
C       linetype     line type (SOLID, DASHED, DOTTED, DASH-DOTTED)
C      linewidth     line width
C      color          color of polyline
C
C----------------------------------------------------------------+
```

```
        INTEGER*4 index, i
        CHARACTER*5 L$, C$
        CHARACTER*3 ascii_x, ascii_y

        REAL*4 ratio
        INTEGER*4 linetype, color
        REAL*4 linewidth
C       Determine the index for the bundle table.
        index = attrib_array.polyline_index
        IF( index .GT. wsl.number_pline_ind ) THEN
           index = 1
        END IF

C-------------------------------------------------------------------+
C     If the attribute is individual, then get it from the individual |
C     attributes list.  Otherwise get it from the bundle table.       |
C-------------------------------------------------------------------+

        IF( BTEST( asf_mask, 0)) THEN
           linetype = attrib_array.line_type
        ELSE
           linetype = wsl.real_pline_bundles( index ).line_type
        END IF

        IF( BTEST( asf_mask, 1)) THEN
           linewidth = attrib_array.linewidth_scale_factor
        ELSE
           linewidth =
     +       wsl.real_pline_bundles(index).linewidth_scale_factor
        END IF
        IF( BTEST( asf_mask, 2)) THEN
           color = attrib_array.polyline_color_index
        ELSE
           color = wsl.real_pline_bundles( index ).color_index
        END IF

C-------------------------------------------------------------------+
C     Decide on requested line type.                                |
C-------------------------------------------------------------------+

        GO TO (110,120,130,140) linetype

C       W(P1)--SOLID
110     L$ =  'W(P'//CHAR( linetype + 48)//')'
        GO TO 150

C       W(P2)--DASHED
120     L$ =  'W(P'//CHAR( linetype + 48)//')'
        GO TO 150

C       W(P4)--DOTTED
130     L$ =  'W(P'//CHAR( linetype + 49)//')'
        GO TO 150

C       W(P3)--DASH-DOTTED
140     L$ =  'W(P'//CHAR( linetype + 47)//')'
150     CONTINUE

C-------------------------------------------------+
C     Ignore linewidth; we only do linewidth of 1. |
C-------------------------------------------------+
```

```
C-----------------------------------------+
C     Decide on different shades of color. |
C-----------------------------------------+
      IF ( color .GE. 4 ) THEN
          color = 1
      END IF
         C$ = 'W(I'//CHAR(color+48)//')'

C-----------------------------------------------------------+
C     CONVERT THE WORLD COORDINATES TO DEVICE COORDINATES |
C-----------------------------------------------------------+

         CALL CONVERT_WC_TO_DC( wsl.trans_matrix,
     +         wc_x_array, wc_y_array, temp_x_array, temp_y_array,
     +      num_pts)

C-------------------------------------------------------------+
C     CLIPPING SHOULD BE PERFORMED HERE    !!!!!!!!           |
C-------------------------------------------------------------+

C-------------------------------------------------------------+
C     CONVERT THE DEVICE COORDINATES TO EQUIVALENT ASCII CODE |
C-------------------------------------------------------------+

         CALL INT_TO_ASCII( temp_x_array(1), temp_y_array(1),
     +     ascii_x, ascii_y)
      CALL ENTER_GRAPHICS( wsl )
C        Turn on the linetype
         WRITE(wsl.unit_num,503) L$
 503     FORMAT('+',A)

C        Turn on the color
         WRITE(wsl.unit_num,504) C$
 504     FORMAT('+',A)
C        Position the cursor to the first point
         WRITE(wsl.unit_num,502) 'P['//ascii_x//','//ascii_y//']V'
 502     FORMAT('+',A)

C-------------------+
C     Draw the line |
C-------------------+

         DO 10 i = 2, num_pts
C           Convert the device coordinates to equivalent ascii code
            CALL INT_TO_ASCII( temp_x_array(i), temp_y_array(i),
     +         ascii_x, ascii_y)

            WRITE(wsl.unit_num,505) '['//ascii_x//','//ascii_y//']'
 505        FORMAT('+',A)
 10     CONTINUE

C-----------------------------------------------------+
C     Return REGIS attributes to their default values. |
C-----------------------------------------------------+

C        Set writing pattern to a solid line.
         WRITE(wsl.unit_num,506) 'W(P1)'
 506     FORMAT('+',A)

      CALL EXIT_GRAPHICS( wsl )
C        Display surface is not empty.
         wsl.display_empty = GKS$K_NOTEMPTY
```

```
C       Return success status
        HANDLER__POLYLINE = GKS$_SUCCESS
        RETURN
        END

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Text"
C  Input Parameters:
C     wsl           address of local data area allocated for driver
C     text_pos_x    starting text position in WC
C     text_pos_y
C     char_string    actual text character string
C     attrib_array    an array of text attributes
C     asf_mask        a 32-bit bitmask holding the attribute source flags
C
C  Output Parameters:
C     NONE.
C
C  Value Returned:
C     GKS$_SUCCESS      Success
C     GKS$_ERROR_101     Invalid code in string
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__TEXT( wsl, text_pos_x,
     +      text_pos_y, char_string, attrib_array, asf_mask)

      IMPLICIT NONE
      INCLUDE 'sys$library:gksmsgs.for/nolist'
      INCLUDE 'sys$library:gksdefs.for/nolist'
      INCLUDE 'ws_state_list.for/nolist'
      INCLUDE 'indiv_attributes.for/nolist'

C     Parameters
        RECORD /ws_state_list/ wsl
        REAL*4  text_pos_x
      REAL*4  text_pos_y
      CHARACTER*(*) char_string
        RECORD /text_attr/ attrib_array
        INTEGER*4 asf_mask

C-----------------------------------------------------------+
C
C     Declare local variables:
C        index          index into polyline attributes list
C      C$          REGIS command to set color
C      P$          REGIS command to position the cursor
C      status          value returned from library function
C          length          length of the descriptor
C       address     address of the descriptor
C
C------------------------------------------------------------+
```

```
        INTEGER*4 index, i, status, length, address
        INTEGER*4 color
        CHARACTER*1 c
        CHARACTER*5 C$
        CHARACTER*10 P$
        RECORD /text_attr/ unbundle_text
        INTEGER*4  LIB$ANALYZE_SDESC
        INTEGER*4 GKS$SIM_STROKE_TEXT
        EXTERNAL HANDLER__POLYLINE, HANDLER__FILL_AREA

        REAL*4 ratio
C       Determine the index for the bundle table.
        index = attrib_array.text_index
        IF( index .GT. wsl.num_text_ind ) THEN
            index = 1
        END IF

C-------------------+
C     Error check:  |
C-------------------+

        status = LIB$ANALYZE_SDESC( char_string, length, address)
        DO 10 i = 1, length
            c = char_string(i:i)
            IF( LLT(c,' ') .OR.
     +                LGT(c, CHAR(15*16+7)) ) THEN
            HANDLER__TEXT = GKS$_ERROR_101
            RETURN
            END IF
10      CONTINUE

C-----------------------------------------------------------------------+
C     If the attribute is individual, then get it from the individual   |
C     attributes list.  Otherwise get it from the bundle table.         |
C-----------------------------------------------------------------------+

        CALL UNBUNDLE_TEXT_ATTR ( wsl, unbundle_text ,
     +      attrib_array, asf_mask)

     IF ( color .GE. 4 ) THEN
         color = 1
     END IF
        C$ = 'W(I'//CHAR(color+48)//')'

C       Determine the precision.
        GO TO (210, 220, 230) (unbundle_text.precision+1)

C       String precision--NOT IMPLEMENTED
210     GO to 230

C       Character precision--NOT IMPLEMENTED
220     GO TO 230
```

```
C       Stroke precision.
230     status = GKS$SIM_STROKE_TEXT( wsl, text_pos_x,
     +     text_pos_y, char_string, unbundle_text,
     +     HANDLER__POLYLINE, HANDLER__FILL_AREA )

C       Display surface is not empty.
        wsl.display_empty = GKS$K_NOTEMPTY

C       Return success status.
        HANDLER__TEXT = GKS$_SUCCESS

        RETURN
        END
```

D.5 Output Attribute Functions

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Set Polyline Representation"
C  Input Parameters:
C     wsl             address of local data area allocated for driver
C     pline_index     polyline index
C     line_type          type of polyline, one of GKS$K_LINE_TYPE_SOLID,
C                   GKS$K_LINE_TYPE_DASHED, GKS$K_LINE_TYPE_DOTTED,
C                   or GKS$K_LINE_TYPE_DASHED_DOTTED
C     linewidth_scale_factor      linewidth scale factor
C     color_index     color index
C
C  Output Parameters:
C     NONE.
C
C  Value Returned:
C     GKS$_SUCCESS      Success
C     GKS$_ERROR_60     Polyline index is invalid
C     GKS$_ERROR_64     Specified linetype is not supported on this workstation
C     GKS$_ERROR_93     Color index is invalid
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__SET_PLINE_REP( wsl,
     +     pline_index, line_type, linewidth_scale_factor, color_index )

      IMPLICIT NONE
      INCLUDE 'sys$library:gksdefs.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'
      INCLUDE 'ws_state_list.for/nolist'
      INCLUDE 'wsdt.for/nolist'

C      Parameters
        RECORD /ws_state_list/ wsl
        INTEGER*4 pline_index
        INTEGER*4 line_type
        REAL*4    linewidth_scale_factor
        INTEGER*4 color_index
      RECORD / wsdt_struct / wsdt
      COMMON / workdesc / wsdt
```

```
C--------------------+
C     Error checking. |
C--------------------+

        IF( ( pline_index .LE. 0) .OR.
     +      (pline_index .GT. wsl.number_pline_ind ) ) THEN
           HANDLER__SET_PLINE_REP = GKS$_ERROR_60
           RETURN
        END IF

     IF ( ( line_type .ne. GKS$K_LINETYPE_SOLID ) .AND.
     +          ( line_type .ne. GKS$K_LINETYPE_DASHED ) .AND.
     +       ( line_type .ne. GKS$K_LINETYPE_DOTTED ) .AND.
     +       ( line_type .ne. GKS$K_LINETYPE_DASHED_DOTTED ) ) THEN
          HANDLER__SET_PLINE_REP = GKS$_ERROR_64
          RETURN
        END IF

        IF( ( color_index .LT. 0 ) .OR.
     +      ( color_index .GE. wsdt.num_colors) ) THEN
          HANDLER__SET_PLINE_REP = GKS$_ERROR_93
          RETURN
        END IF
C-----------------------------+
C     Set the polyline bundle. |
C-----------------------------+

C     Put in the SET values
        wsl.set_pline_bundles(pline_index).line_type = line_type
        wsl.set_pline_bundles(pline_index).linewidth_scale_factor =
     +      linewidth_scale_factor
        wsl.set_pline_bundles(pline_index).color_index = color_index

C     Put in the REALIZED values
        wsl.real_pline_bundles(pline_index).line_type = line_type
        wsl.real_pline_bundles(pline_index).linewidth_scale_factor =
     +                                  1.0
        wsl.real_pline_bundles(pline_index).color_index = color_index

C       Return status of success.
        HANDLER__SET_PLINE_REP = GKS$_SUCCESS

        RETURN
        END
```

D.6 Inquiry Functions

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Inquire List of Polyline Indices"
C  Input Parameters:
C     wsl              address of local data area allocated for driver
C
C  Modified Parameters:
C     num_pline_entries     (I)number of entries available in the array;
C               (O)number of entries written in the array
C
C  Output Parameters:
C     list_pline_ind      list of defined polyline indices
C     total_num_ind     total number of indices available
C
C  Value Returned:
C     GKS$_SUCCESS     Success
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__INQ_PLINE_IND( wsl,
     +      num_pline_entries, list_pline_ind, total_num_ind )

     IMPLICIT NONE
     INCLUDE 'sys$library:gksmsgs.for/nolist'
     INCLUDE 'sys$library:gksdefs.for/nolist'
     INCLUDE 'ws_state_list.for/nolist'

C     Parameters
        RECORD /ws_state_list/ wsl
        INTEGER*4 num_pline_entries
        INTEGER*4 list_pline_ind(*)
        INTEGER*4 total_num_ind
     INTEGER*4 i

     IF ( num_pline_entries .GT. wsl.number_pline_ind)  THEN
         num_pline_entries = wsl.number_pline_ind
     END IF
     total_num_ind = wsl.number_pline_ind
     DO 100 i = 1,num_pline_entries
         list_pline_ind(i) = wsl.set_pline_bundles(i).pline_index
 100     CONTINUE
C       Return status of success.
     HANDLER__INQ_PLINE_IND = GKS$_SUCCESS
        RETURN
        END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Inquire Polyline Representation"
C  Input Parameters:
C     wsl              address of local data area allocated for driver
C     pline_index      polyline index
C     set_realized      returned value is GKS$K_VALUE_SET
C                 or GKS$K_VALUE_REALIZED
C
C  Output Parameters:
C     line_type              line type in the bundle
C     linewidth_scale_factor      line width scale factor in the bundle
C     color_index      polyline color index
C
C  Value Returned:
C     GKS$_SUCCESS      Success
C     GKS$_ERROR_60      Polyline index is invalid
C     GKS$_ERROR_61      A representation for the specified polyline index has not
C                been defined on this workstation
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

         INTEGER*4 FUNCTION HANDLER__INQ_PLINE_REP( wsl, pline_index,
      +      set_realized, line_type, linewidth_scale_factor,
      +      color_index )

      IMPLICIT NONE
      INCLUDE 'sys$library:gksdefs.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'
      INCLUDE 'ws_state_list.for/nolist'

C      Parameters

         RECORD /ws_state_list/ wsl
         INTEGER*4 pline_index
         INTEGER*4 set_realized
         INTEGER*4 line_type
         REAL*4 linewidth_scale_factor
         INTEGER*4 color_index
C      Error checking
      IF ( ( pline_index .LT. 0 ) .OR.
      +       ( pline_index .GT. wsl.number_pline_ind ) ) THEN
          HANDLER__INQ_PLINE_REP = GKS$_ERROR_60
          RETURN
      END IF

      IF ( set_realized .EQ. GKS$K_VALUE_SET ) THEN
          line_type = wsl.set_pline_bundles(pline_index).line_type
          linewidth_scale_factor =
      +        wsl.set_pline_bundles(pline_index).linewidth_scale_factor
          color_index =
      +        wsl.set_pline_bundles(pline_index).color_index
      ELSE
          line_type = wsl.real_pline_bundles(pline_index).line_type
          linewidth_scale_factor =
      +      wsl.real_pline_bundles(pline_index).linewidth_scale_factor
          color_index =
      +        wsl.real_pline_bundles(pline_index).color_index
      END IF
```

```
C       Return status of success.
     HANDLER__INQ_PLINE_REP = GKS$_SUCCESS
        RETURN
        END


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Inquire Polyline Facilities"
C  Input Parameters:
C     wstype               workstation type
C
C  Modified Parameters:
C     num_ltypes    (I)number of available elements in the array;
C                   (O)number of elements written in the array
C
C  Output Parameters:
C     total_num_ltypes  total number of linetypes
C     list_line_types       list of available linetypes
C     num_line_widths     number of available linewidths
C     nominal_linewidth     nominal linewidth (LDC)
C     minimum_linewidth     minimum linewidth (LDC)
C     maximim_linewidth maximum linewidth (LDC)
C     number_predef_pline_ind  number of predefined polyline indices
C
C  Value Returned:
C     GKS$_SUCCESS     Success
c
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__INQ_PLINE_FAC( wstype, num_ltypes,
     +      total_num_ltypes, list_line_types, num_linewidths,
     +      nominal_linewidth, minimum_linewidth, maximum_linewidth,
     +      number_predef_pline_ind )

     IMPLICIT NONE
     INCLUDE 'sys$library:gksdefs.for/nolist'
     INCLUDE 'sys$library:gksmsgs.for/nolist'
     INCLUDE 'wsdt.for/nolist'
C     Parameters

        INTEGER*4 wstype
        INTEGER*4 num_ltypes
        INTEGER*4 total_num_ltypes
        INTEGER*4 list_line_types(*)
        INTEGER*4 num_linewidths
        REAL*4 nominal_linewidth
        REAL*4 minimum_linewidth
        REAL*4 maximum_linewidth
        INTEGER*4 number_predef_pline_ind
      INTEGER*4 i
```

```
        RECORD /wsdt_struct/ wsdt
      COMMON /workdesc/ wsdt
C      Make sure that the bundle tables are initialized first.
      CALL BUNDLE_INIT()
      IF ( num_ltypes .GT. wsdt.num_linetypes ) THEN
          num_ltypes = wsdt.num_linetypes
      END IF
        total_num_ltypes = wsdt.num_linetypes
        DO 100 i = 1, num_ltypes
           list_line_types(i) = wsdt.list_line_types(i)
 100    CONTINUE

        num_linewidths = wsdt.num_linewidths
        nominal_linewidth = wsdt.nominal_linewidth
        minimum_linewidth = wsdt.minimum_linewidth
        maximum_linewidth = wsdt.maximum_linewidth
        number_predef_pline_ind = wsdt.number_predef_pline_ind

C       Return status of success.
        HANDLER__INQ_PLINE_FAC = GKS$_SUCCESS

        RETURN
        END


CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  "Inquire Predefined Polyline Representation"
C  Input Parameters:
C     wstype          workstation type
C     pline_index     predefined polyline index
C
C  Output Parameters:
C     linetype          linetype
C     linewidth          linewidth scale factor
C     color_index     polyline color index
C
C  Value Returned:
C     line_type          linetype
C     line_width_scale_factor      linewidth scale factor
C     color_index     polyline color index
C
C  Value Returned:
C     GKS$_SUCCESS     Success
C     GKS$_ERROR_60     Polyline index is invalid
C     GKS$_ERROR_62     A representation for the specified polyline index has not
C                     been predefined on this workstation
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        INTEGER*4 FUNCTION HANDLER__INQ_PREDEF_PLINE_REP( wstype,
     +         pline_index, line_type, line_width_scale_factor,
     +      color_index)

      IMPLICIT NONE
      INCLUDE 'sys$library:gksdefs.for/nolist'
      INCLUDE 'sys$library:gksmsgs.for/nolist'
      INCLUDE 'wsdt.for/nolist'
```

```
C       Parameters
        INTEGER*4 wstype
        INTEGER*4 pline_index
        INTEGER*4 line_type
        REAL*4    line_width_scale_factor
        INTEGER*4 color_index
        RECORD /wsdt_struct/ wsdt
        COMMON /workdesc/ wsdt
C     Make sure that the bundle tables are initialized first.
      CALL BUNDLE_INIT()
      IF (pline_index .GT. wsdt.number_predef_pline_ind) THEN
          HANDLER__INQ_PREDEF_PLINE_REP = GKS$_ERROR_62
          RETURN
      END IF
      line_type  = wsdt.pline_bundles(pline_index).line_type
      line_width_scale_factor =
     +       wsdt.pline_bundles(pline_index).linewidth_scale_factor
      color_index = wsdt.pline_bundles(pline_index).color_index
C        Return status of success.
      HANDLER__INQ_PREDEF_PLINE_REP = GKS$_SUCCESS
        RETURN
        END
```

D.7 DFT Building Macro

```
;
; Create a DFT for the FORTRAN VT handler.
;

     .TITLE FORTRAN_DFT

;FORTRAN_DFT_ADDR::
```

```
DFT -
HANDLER = FORTRAN_DFT_ADDR,-
WS_CAT = OUT,-
OPEN_WS = HANDLER__OPEN_WS,-
CLEAR_WS = HANDLER__CLEAR_WS,-
SET_DEFER = HANDLER__SET_DEFERRAL, -
PERFORM_DEFERRED = HANDLER__PERFORM_DEFERRED, -
 SET_GLOBAL = HANDLER__SET_GLOBAL, -
HIGHLIGHT_EXT = HANDLER__HIGHLIGHT_EXTENT, -
MSG = HANDLER__MESSAGE, -
CLOSE_WS = HANDLER__CLOSE_WS, -
SET_NDC_XFORM = HANDLER__SET_NDC_XFORM, -
ESC = HANDLER__ESCAPE,-
PLINE = HANDLER__POLYLINE,-
PMARKER = HANDLER__POLYMARKER,-
FILL_AREA = HANDLER__FILL_AREA,-
TXT = HANDLER__TEXT,-
CELL_ARRAY = HANDLER__CELL_ARRAY,-
GDP = HANDLER__GDP,-
-
SET_PLINE_REP = HANDLER__SET_PLINE_REP,-
SET_PMARK_REP = HANDLER__SET_PMARKER_REP,-
SET_TEXT_REP = HANDLER__SET_TEXT_REP,-
SET_FILL_REP = HANDLER__SET_FILL_REP,-
SET_PATT_REP = HANDLER__SET_PATT_REP,-
SET_COLOR_REP = HANDLER__SET_COLOR_REP,-
-
SET_WS_WIND = HANDLER__SET_WS_WINDOW,-
SET_WS_VIEW = HANDLER__SET_WS_VIEWPORT,-
SET_NORM_XFORM = HANDLER__SET_NORM_XFORM

DFT_GKS_INQ-
INQ_WS_DEFER = HANDLER__INQ_WS_DEF_UPDATE,-
INQ_LIST_PLINE = HANDLER__INQ_PLINE_IND,-
INQ_PLINE_REP = HANDLER__INQ_PLINE_REP,-
INQ_LIST_PMARK = HANDLER__INQ_PMARK_IND,-
INQ_PMARK_REP = HANDLER__INQ_PMARK_REP,-
INQ_LIST_TEXT = HANDLER__INQ_TEXT_IND,-
INQ_TEXT_REP = HANDLER__INQ_TEXT_REP,-
INQ_TEXT_EXTENT = HANDLER__INQ_TEXT_EXTENT,-
INQ_LIST_FILL = HANDLER__INQ_FILL_IND,-
INQ_FILL_REP = HANDLER__INQ_FILL_REP,-
INQ_LIST_PATT = HANDLER__INQ_PATT_IND,-
INQ_PATT_REP = HANDLER__INQ_PATT_REP,-
INQ_LIST_COLOR = HANDLER__INQ_COLOR_IND,-
INQ_COLOR_REP = HANDLER__INQ_COLOR_REP,-
INQ_WS_XFORM = HANDLER__INQ_WS_XFORM,-
INQ_PIXEL = HANDLER__INQ_PIXEL,-
INQ_PIXEL_DIMEN = HANDLER__INQ_PIXEL_ARRAY_DIM,-
INQ_PIXEL_ARRAY = HANDLER__INQ_PIXEL_ARRAY
```

```
    DFT_WS_INQ -
    INQ_WS_CAT = HANDLER__INQ_WS_CAT,-
    INQ_WS_CLASS = HANDLER__INQ_WS_CLASS,-
    INQ_MAX_DISP = HANDLER__INQ_DISPLAY_SIZE,-
    INQ_PLINE_FAC = HANDLER__INQ_PLINE_FAC,-
    INQ_PREDEF_PLINE = HANDLER__INQ_PREDEF_PLINE_REP,-
    INQ_PMARK_FAC = HANDLER__INQ_PMARK_FAC,-
    INQ_PREDEF_PMARK = HANDLER__INQ_PREDEF_PMARK_REP,-
    INQ_TEXT_FAC = HANDLER__INQ_TEXT_FAC,-
    INQ_PREDEF_TEXT = HANDLER__INQ_PREDEF_TEXT_REP,-
    INQ_FILL_FAC = HANDLER__INQ_FILL_FAC,-
    INQ_PREDEF_FILL = HANDLER__INQ_PREDEF_FILL_REP,-
    INQ_PATT_FAC = HANDLER__INQ_PATT_FAC,-
    INQ_PREDEF_PATT = HANDLER__INQ_PREDEF_PATT_REP,-
    INQ_COLOR_FAC = HANDLER__INQ_COLOR_FAC,-
    INQ_PREDEF_COLOR = HANDLER__INQ_PREDEF_COLOR_REP,-
    INQ_AVAIL_GDP = HANDLER__INQ_LIST_GDP,-
    INQ_GDP = HANDLER__INQ_GDP,-
    INQ_DYN_MOD_WS = HANDLER__INQ_DYN_MOD_WS_ATTR,-
    INQ_DYN_SEG_ATTR = HANDLER__INQ_DYN_MOD_SEG_ATTR,-
    INQ_DFLT_DEFER = HANDLER__INQ_DEF_DEF_STATE,-
    INQ_MAX_LEN_STATE_TABLE = HANDLER__INQ_MAX_WS_STATE, -

    INQ_STORAGE_SIZE = HANDLER__INQ_SIZE_STORAGE
.END
```

D.8 Linking Command Procedure

```
$ !
$ ! This command procedure makes the shareable image for the handler
$ ! and then defines the logicals needed to run it.
$ !
$ ! It is linked "/debug" to allow for shareable image debugging
$ !
$ ! The WORKDESC common attributes must be redefined so that they
$ ! area NOSHR,WRT
$ !
$ link/debug/share=ws_handler.exe/map/full/cross fortran_dft,sys$input/opt
    handler/lib
    universal = fortran_dft_addr
    PSECT_ATTR=WORKDESC,NOSHR,WRT
```

```
$ !
$ ! Use workstation type 110, but this can be any unused number.
$ !
$ def gks$wstype 110
$ def gks$workstation_110 ws_handler
$ def ws_handler disk$:[your_directory]ws_handler.exe
$ def gks$function_tab_110 fortran_dft_addr
$ !
$ ! workstation is non-reentrant so define a logical.
$ ! NOTE: - when debugging don't define this logical as the image
$ ! name will be changed.
$ !
$ ! def gks$non_reentrant true
$ !
$ ! Define the logical GKS$$HANDLER_DEBUG to be anything so that the
$ ! debugger will wake up in open workstation.
$ def gks$$handler_debug T
```

Index

A

Attributes, 2–4
  geometric, 9–1 to 9–2
  nongeometric, 9–2
Attribute Source Flag, 2–4

B

Bundles, 2–4

C

Cell Array function, 9–19 to 9–21
CHOICE_DATAREC
  in WSL, 3–30
CHOICE_DATA_RECORD, 3–23
Clear Workstation function, 4–7 to 4–8
Close Segment function, 10–3
Close Workstation function, 4–6
COLOR_AVAILABLE, 3–17
COLOR_BUNDLES, 3–27
Communications
  between server and handler, 2–5 to 2–7
CONNECTION_ID, 3–26
Coordinate systems, 2–3
  device coordinates, 2–3
  Logical device coordinates, 2–3
  mapping, 2–3
    from DC to LDC, 2–3
    from NDC to DC, 2–4
    from WC to NDC, 2–3
  normalized device coordinates, 2–3
  table, 2–3
  world coordinates, 2–3
Create Segment function, 10–2
CUR_WS_VIEWPORT, 3–27
CUR_WS_WINDOW, 3–27

D

Data Structures
  for workstation handlers, 3–4 to 3–31
Data types, 2–6
  arrays
    one-dimensional, 2–6
    two-dimensional, 2–6
  integer, 2–6
  real numbers, 2–6
  table, 2–6
  text, 2–6
DC
  See Device coordinates
DEFER_MODE, 3–26
DEF_DEFER_MODE, 3–9
Delete Segment function, 10–5 to 10–6
Description Table
  GKS, 2–2
  Workstation, 2–2
Device coordinates, 2–3
DEVNUM, 3–21, 3–22, 3–23, 3–24
  in WSL, 3–28, 3–29, 3–30, 3–31
DEV_COORDINATE_UNITS, 3–6
DEV_DISPLAY_SPACE_SIZE_X, 3–6
DEV_DISPLAY_SPACE_SIZE_Y, 3–6
DISPLAY_EMPTY, 3–26
DISPLAY_TYPE, 3–6
DMAF_COLOR, 3–8
DMAF_DELETE_SEGMENT, 3–21
DMAF_FILL, 3–8
DMAF_HIGHLIGHTING, 3–20
DMAF_INVISIBILITY, 3–19

DMAF_PATTERN, 3–8
DMAF_POLYLINE, 3–7
DMAF_POLYMARKER, 3–7
DMAF_SEGMENT_OVERLAP, 3–20
DMAF_SEGMENT_PRIORITY, 3–20
DMAF_SEGMENT_XFORM, 3–19
DMAF_TEXT, 3–7
DMAF_VISIBILITY, 3–20
DMAF_WS_TRANSFORMATION, 3–9

E

ECHO_AREA, 3–21, 3–22, 3–23, 3–24
  in WSL, 3–28, 3–29, 3–30, 3–31
ECHO_SWITCH
  in WSL, 3–28, 3–29, 3–30, 3–31
Escape function, 4–12 to 4–13

F

Fill Area function, 9–15 to 9–18
FILL_BUNDLES, 3–27
FLAG_PRESENT, 3–26

G

GDP function, 9–22 to 9–23
Get Item from Metafile function, 7–5 to 7–6
GKS$FIND_SEGMENT, C–2 to C–5
GKS$FIND_SEG_EXTENT, C–6 to C–7
GKS$NON_REENTRANT_nn, 3–48
GKS$SIM_STROKE_INQ_TEXT_FAC, B–9 to B–10
GKS$SIM_STROKE_TEXT, B–2 to B–5
GKS$SIM_STROKE_TEXT_EXTENT, B–6 to B–8
GKSDEFS, 3–33
  file extensions for different languages, 3–33
GKS Description Table, 2–2
GKS Kernel, 1–1
GKSMSGS, 3–33
  file extensions for different languages, 3–33
GKS State List, 2–2

H

Handlers, 1–1
Hard-copy output
  in workstation handlers, 3–3
Highlight Extent function, 9–24

I

Initialize Choice function, 5–21 to 5–24
Initialize Locator function, 5–7 to 5–12
Initialize Pick function, 5–28 to 5–30
Initialize String function, 5–25 to 5–27
Initialize Stroke function, 5–13 to 5–17
Initialize Valuator function, 5–18 to 5–20
INITIAL_PICKID
  in WSL, 3–31
INITX_ARRAY
  in WSL, 3–29
INITY_ARRAY
  in WSL, 3–29
INIT_CHOICE
  in WSL, 3–30
INIT_LOCN_X, 3–21
  in WSL, 3–28
INIT_LOCN_Y, 3–21
  in WSL, 3–28
INIT_SEGMENT
  in WSL, 3–31
INIT_STATUS
  in WSL, 3–30, 3–31
INIT_STRING
  in WSL, 3–31
INIT_VALUE, 3–23
  in WSL, 3–30
Input, 2–4
  for device handlers, 2–4
  logical input types, 2–4
  transformations in, 2–4
Inquire Choice Device State function, 6–47 to 6–49
Inquire Color Facilities function, 6–90
Inquire Color Representation function, 6–35 to 6–36
Inquire Default Choice Device Data function, 6–106 to 6–107
Inquire Default Deferral State Values function, 6–114 to 6–115
Inquire Default Locator Device Data function, 6–100 to 6–101
Inquire Default Pick Device Data function, 6–110 to 6–111
Inquire Default String Device Data function, 6–108 to 6–109
Inquire Default Stroke Device Data function, 6–102 to 6–103
Inquire Default Valuator Device Data function, 6–104 to 6–105

Inquire Display Space Size function, 6–69 to 6–70
Inquire Dynamic Modification of Segment Attributes function, 6–117 to 6–118
Inquire Dynamic Modification of Workstation Attributes function, 6–112 to 6–113
Inquire Fill Area Facilities function, 6–83 to 6–84
Inquire Fill Area Representation function, 6–27 to 6–28
INQUIRE GDP EXTENT, 3–34 to 3–35
Inquire GDP Primitives function, 6–92 to 6–93
Inquire Generalized Drawing Primitive function, 6–94 to 6–95
Inquire List of Color Indexes function, 6–33 to 6–34
Inquire List of Fill Area Indexes function, 6–25 to 6–26
Inquire List of Pattern Indexes function, 6–29 to 6–30
Inquire List of Polyline Indexes function, 6–3 to 6–4
Inquire List of Polymarker Indexes function, 6–7 to 6–8
Inquire List of Text Indexes function, 6–11 to 6–12
Inquire Locator Device State function, 6–39 to 6–41
Inquire Maximum Length of Workstation State Tables function, 6–96 to 6–97
Inquire Number of Available Logical Input Devices function, 6–98 to 6–99
Inquire Number of Segment Priorities Supported function, 6–116
Inquire Pattern Facilities function, 6–87
Inquire Pattern Representation function, 6–31 to 6–32
Inquire Pick Device State function, 6–52 to 6–54
Inquire Pixel Array Dimensions function, 6–57 to 6–59
Inquire Pixel Array function, 6–60 to 6–61
Inquire Pixel function, 6–62 to 6–63
Inquire Polyline Facilities function, 6–71 to 6–72
Inquire Polyline Representation function, 6–5 to 6–6
Inquire Polymarker Facilities function, 6–75 to 6–76
Inquire Polymarker Representation function, 6–9 to 6–10
Inquire Predefined Color Representation function, 6–91
Inquire Predefined Fill Area Representation function, 6–85 to 6–86
Inquire Predefined Pattern Representation function, 6–88 to 6–89
Inquire Predefined Polyline Representation function, 6–73 to 6–74
Inquire Predefined Polymarker Representation function, 6–77 to 6–78
Inquire Predefined Text Representation function, 6–81 to 6–82
Inquire Segment Names on Workstation function, 6–64 to 6–65
inquire Size of Handler Storage function, 6–119
Inquire Size of Handler Storage function, 6–119
Inquire String Device State function, 6–50 to 6–51
Inquire Stroke Device State function, 6–42 to 6–44
Inquire Text Extent function, 6–15 to 6–24
Inquire Text Facilities function, 6–79 to 6–80
Inquire Text Representation function, 6–13 to 6–14
Inquire Valuator Device State function, 6–45 to 6–46
Inquire Workstation Category function, 6–66 to 6–67
Inquire Workstation Classification function, 6–68
Inquire Workstation Deferral and Update State function, 6–55 to 6–56
Inquire Workstation Transformation function, 6–37 to 6–38
Inquiries, 2–1 to 2–2
Insert Segment
  and the NDC transformation, A–2

K

Kernel, 1–1

L

LDC
  See Logical device coordinates
LIST_FILL_INTSTYLE, 3–15
LIST_FONT_PREC_PAIRS, 3–13
LIST_GDP, 3–18
LIST_HATCH_STYLE, 3–15
LIST_LINE_TYPES, 3–10
LIST_MARKERTYPES, 3–12
LIST_PROMPT_ECHO_TYPES, 3–21, 3–22, 3–23, 3–24
LOCATOR_DATA_RECORD, 3–21
LOC_DATAREC
  in WSL, 3–28
Logical device coordinates, 2–3

M

MAXIMUM_BUFSIZE, 3–22, 3–24
MAXIMUM_CHAR_EXP_FACTOR, 3–14
MAXIMUM_CHAR_HEIGHT, 3–14
MAXIMUM_LINEWIDTH, 3–11
MAXIMUM_MSIZE, 3–12
MAX_COLOR_IND, 3–19
MAX_FILL_BUNDLES, 3–19
MAX_NUM_CHOICE, 3–23
MAX_PATT_IND, 3–19
MAX_PLINE_BUNDLES, 3–18
MAX_PMARK_BUNDLES, 3–19
MAX_TEXT_BUNDLES, 3–19
Message function, 4–25
MINIMUM_CHAR_EXP_FACTOR, 3–14
MINIMUM_CHAR_HEIGHT, 3–14
MINIMUM_LINEWIDTH, 3–10
MINIMUM_MSIZE, 3–12

N

NDC
  See Normalized device coordinates
NDC transformation, A–2 to A–6
NEW_FRAME, 3–26
NOMINAL_LINEWIDTH, 3–10
NOMINAL_MSIZE, 3–12
Normalization transformation, 2–3
Normalized device coordinates, 2–3
NUMBER_PLINE_IND, 3–26
NUMBER_PREDEF_PLINE_IND, 3–11
NUMBER_PREDEF_PMARK_IND, 3–12
NUM_CHAR_EXP_FACTORS, 3–14
NUM_CHAR_HEIGHTS, 3–14
NUM_COLORS, 3–17
NUM_COLOR_REP, 3–27
NUM_FILL_IND, 3–27
NUM_FILL_INTSTYLE, 3–15
NUM_FONT_PREC_PAIRS, 3–13
NUM_GDP, 3–17
NUM_HATCH_STYLE, 3–15
NUM_INIT_POINTS
  in WSL, 3–29
NUM_LINEWIDTHS, 3–10
NUM_LINE_TYPES, 3–10
NUM_MARKERTYPES, 3–11
NUM_MSIZES, 3–12
NUM_PATT_IND, 3–27
NUM_PREDEF_COLOR_REP, 3–17
NUM_PREDEF_FILL_IND, 3–15
NUM_PREDEF_PATT_IND, 3–16
NUM_PREDEF_TEXT_IND, 3–14
NUM_PREDER_PMAR_IND, 3–26
NUM_PROMPT_ECHO_TYPES, 3–21, 3–22, 3–23, 3–24
NUM_SEGMENT_PRIORITIES, 3–19
NUM_TEXT_IND, 3–27

O

Open Workstation
  initializing the WSL, 3–25
Open Workstation function, 4–4 to 4–5
OPMODE
  in WSL, 3–28, 3–29, 3–30, 3–31
Output, 2–4 to 2–6
  and overlapping segments, 9–5
  considerations in writing a handler, 2–5
  coordinate data, 9–1
  geometric attributes, 9–1 to 9–2
  nongeometric attributes, 9–2
  simulating output functions, 2–5
  transformations in, 2–5

P

Passing mechanisms, 2–6 to 2–7
  table, 2–6
PATT_BUNDLES, 3–27
Perform Deferred Output function, 4–11
PICK_DATAREC
  in WSL, 3–31
PICK_DATA_RECORD, 3–24
PLINE_BUNDLES, 3–26
PMARK_BUNDLES, 3–27
Polyline function, 9–6 to 9–7
Polymarker function, 9–8 to 9–9
PREDEF_COLOR_REPS, 3–17
PREDEF_FILL_BUNDLES, 3–16
PREDEF_PATT_REPS, 3–16
PREDEF_PLINE_BUNDLES, 3–11
PREDEF_PMARK_BUNDLES, 3–13
PREDEF_TEXT_BUNDLES, 3–15
Primitives, 2–4
PROMPT_ECHO_TYPE
  in WSL, 3–28, 3–29, 3–30, 3–31

R

RASTER_DISPLAY_SPACE_SIZE_X, 3–6
RASTER_DISPLAY_SPACE_SIZE_Y, 3–6

Read Item from Metafile function, 7–7 to 7–8
Redraw All Segments on Workstation function, 4–22
Reentrant handlers, 3–48
REGEN_MODE, 3–9, 3–26
Rename Segment function, 10–4
Request Choice function, 5–49 to 5–50
Request Locator function, 5–43 to 5–44
Request Pick function, 5–53 to 5–54
Request String function, 5–51 to 5–52
Request Stroke function, 5–45 to 5–46
Request Valuator function, 5–47 to 5–48
REQ_WS_VIEWPORT, 3–27
REQ_WS_WINDOW, 3–27

S

Sample Choice function, 5–58 to 5–59
Sample Locator function, 5–55
Sample Pick function, 5–63 to 5–64
Sample String function, 5–61 to 5–62
Sample Stroke function, 5–56 to 5–57
Sample Valuator function, 5–60
Segment State List, 2–2
Segment transformation, A–2
Set Choice Mode function, 5–37 to 5–38
Set Color Representation function, 8–13
Set Deferral Mode function, 4–20 to 4–21
Set Detectability function, 10–13
Set Fill Area Representation function, 8–9 to 8–10
Set Global Interactions function, 4–23 to 4–24
Set Highlighting function, 10–14 to 10–15
Set Locator Mode function, 5–31 to 5–32
Set NDC Transformation function, 4–26
Set Normalization Transformation function, 4–18 to 4–19
Set Pattern Representation function, 8–11 to 8–12
Set Pick Mode function, 5–41 to 5–42
Set Polyline Representation function, 8–3 to 8–4
Set Polymarker Representation function, 8–5 to 8–6
Set Segment Priority function, 10–11 to 10–12
Set Segment Transformation
  and the NDC transformation, A–4
Set Segment Transformation function, 10–7 to 10–8
Set String Mode function, 5–39 to 5–40
Set Stroke Mode function, 5–33 to 5–34
Set Text Representation function, 8–7 to 8–8
Set Valuator Mode function, 5–35 to 5–36
Set Visibility function, 10–9 to 10–10
Set Workstation Viewport function, 4–16 to 4–17
Set Workstation Window function, 4–14 to 4–15
State List
  GKS, 2–2
  Segment, 2–2
  Workstation, 2–2
STK_DATAREC
  in WSL, 3–29
STORED_SEGMENTS, 3–26
STRING_DATAREC
  in WSL, 3–31
STRING_DATA_RECORD, 3–25
Stroke-precision text simulation
  and Pascal Bound Procedure Value, B–1
  implementing, B–1
STROKE_DATA_RECORD, 3–22

T

Text function, 9–10 to 9–14
TEXT_BUNDLES, 3–27
Transformation matrix
  combining, A–1
Transformation pipeline
  for handlers that simulate segments
    figure, A–2
  for handlers that support segments
    figure, A–4
Transformations, 2–2 to 2–4
  derivation
    assuming the identity NDC transformation, A–6 to A–7
    assuming the non-identity NDC transformation, A–8 to A–10
  NDC, A–2
  Segment, A–2
  workstation, 2–4
TRANSFORM_FLAG, 3–26

U

Update Workstation function, 4–9 to 4–10

V

VALUATOR_DATA_RECORD, 3–23
VAL_DATAREC
  in WSL, 3–30

W

WC
  See World coordinates
Worksation State List
  for workstation handlers
    initial values, 3–25
Workstation Description Table, 2–2
  for workstation handlers, 3–4 to 3–25
    contents, 3–4
    contents for CHOICE input devices, 3–23
    contents for LOCATOR input devices, 3–21
    contents for PICK input devices, 3–24
    contents for STRING input devices, 3–24
    contents for STROKE input devices, 3–22
    contents for VALUATOR input devices, 3–22
    for all workstation types except MI and MO, table, 3–5
    for INPUT and OUTIN workstations, 3–21 to 3–25
    for OUTPUT and OUTIN workstations, 3–6 to 3–21
    Table, 3–5
Workstation handler, 1–3
  advantages, 1–4
  figure, 1–3
  hard copy output, 3–3
  Open Workstation
    initializing the WSL, 3–25
  required capabilities, 3–2 to 3–3
    table, 3–2
Workstation Handler escape functions
  INQUIRE GDP EXTENT, 3–34 to 3–35
Workstation handler functions
  Cell Array, 9–19 to 9–21
  Clear Workstation, 4–7 to 4–8
  Close Segment, 10–3
  Close Workstation, 4–6
  Create Segment, 10–2
  Delete Segment, 10–5 to 10–6
  Escape, 4–12 to 4–13
  Fill Area, 9–15 to 9–18
  GDP, 9–22 to 9–23
  Get Item from Metafile, 7–5 to 7–6
  Highlight Extent, 9–24
  Initialize Choice, 5–21 to 5–24
  Initialize Locator, 5–7 to 5–12
  Initialize Pick, 5–28 to 5–30
  Initialize String, 5–25 to 5–27
  Initialize Stroke, 5–13 to 5–17
Workstation handler functions (cont'd.)
  Initialize Valuator, 5–18 to 5–20
  Inquire Choice Device State, 6–47 to 6–49
  Inquire Color Facilities, 6–90
  Inquire Color Representation, 6–35 to 6–36
  Inquire Default Choice Device Data, 6–106 to 6–107
  Inquire Default Deferral State Values, 6–114 to 6–115
  Inquire Default Locator Device Data, 6–100 to 6–101
  Inquire Default Pick Device Data, 6–110 to 6–111
  Inquire Default String Device Data, 6–108 to 6–109
  Inquire Default Stroke Device Data, 6–102 to 6–103
  Inquire Default Valuator Device Data, 6–104 to 6–105
  Inquire Display Space Size, 6–69 to 6–70
  Inquire Dynamic Modification of Segment Attributes, 6–117 to 6–118
  Inquire Dynamic Modification of Workstation Attributes, 6–112 to 6–113
  Inquire Fill Area Facilities, 6–83 to 6–84
  Inquire Fill Area Representation, 6–27 to 6–28
  Inquire GDP Primitives, 6–92 to 6–93
  Inquire Generalized Drawing Primitive, 6–94 to 6–95
  Inquire List of Color Indexes, 6–33 to 6–34
  Inquire List of Fill Area Indexes, 6–25 to 6–26
  Inquire List of Pattern Indexes, 6–29 to 6–30
  Inquire List of Polyline Indexes, 6–3 to 6–4
  Inquire List of Polymarker Indexes, 6–7 to 6–8
  Inquire List of Text Indexes, 6–11 to 6–12
  Inquire Locator Device State, 6–39 to 6–41
  Inquire Maximum Length of Workstation State Tables, 6–96 to 6–97
  Inquire Number of Available Logical Input Devices, 6–98 to 6–99
  Inquire Number of Segment Priorities Supported, 6–116
  Inquire Pattern Facilities, 6–87
  Inquire Pattern Representation, 6–31 to 6–32
  Inquire Pick Device State, 6–52 to 6–54
  Inquire Pixel, 6–62 to 6–63
  Inquire Pixel Array, 6–60 to 6–61
  Inquire Pixel Array Dimensions, 6–57 to 6–59
  Inquire Polyline Facilities, 6–71 to 6–72
  Inquire Polyline Representation, 6–5 to 6–6
  Inquire Polymarker Facilities, 6–75 to 6–76

Workstation handler functions (cont'd.)
Inquire Polymarker Representation, 6–9 to 6–10
Inquire Predefined Color Representation, 6–91
Inquire Predefined Fill Area Representation, 6–85 to 6–86
Inquire Predefined Pattern Representation, 6–88 to 6–89
Inquire Predefined Polyline Representation, 6–73 to 6–74
Inquire Predefined Polymarker Representation, 6–77 to 6–78
Inquire Predefined Text Representation, 6–81 to 6–82
Inquire Segment Names on Workstation, 6–64 to 6–65
inquire Size of Handler Storage, 6–119
Inquire String Device State, 6–50 to 6–51
Inquire Stroke Device State, 6–42 to 6–44
Inquire Text Extent, 6–15 to 6–24
Inquire Text Facilities, 6–79 to 6–80
Inquire Text Representation, 6–13 to 6–14
Inquire Valuator Device State, 6–45 to 6–46
Inquire Workstation Category, 6–66 to 6–67
Inquire Workstation Classification, 6–68
Inquire Workstation Deferral and Update State, 6–55 to 6–56
Inquire Workstation Transformation, 6–37 to 6–38
Message, 4–25
Open Workstation, 4–4 to 4–5
Perform Deferred Output, 4–11
Polyline, 9–6 to 9–7
Polymarker, 9–8 to 9–9
Read Item from Metafile, 7–7 to 7–8
Redraw All Segments on Workstation, 4–22
Rename Segment, 10–4
Request Choice, 5–49 to 5–50
Request Locator, 5–43 to 5–44
Request Pick, 5–53 to 5–54
Request String, 5–51 to 5–52
Request Stroke, 5–45 to 5–46
Request Valuator, 5–47 to 5–48
Sample Choice, 5–58 to 5–59
Sample Locator, 5–55
Sample Pick, 5–63 to 5–64
Sample String, 5–61 to 5–62
Sample Stroke, 5–56 to 5–57
Sample Valuator, 5–60
Set Choice Mode, 5–37 to 5–38
Set Color Representation, 8–13
Workstation handler functions (cont'd.)
Set Deferral Mode, 4–20 to 4–21
Set Detectability, 10–13
Set Fill Area Representation, 8–9 to 8–10
Set Global Interactions, 4–23 to 4–24
Set Highlighting, 10–14 to 10–15
Set Locator Mode, 5–31 to 5–32
Set NDC Transformation, 4–26
Set Normalization Transformation, 4–18 to 4–19
Set Pattern Representation, 8–11 to 8–12
Set Pick Mode, 5–41 to 5–42
Set Polyline Representation, 8–3 to 8–4
Set Polymarker Representation, 8–5 to 8–6
Set Segment Priority, 10–11 to 10–12
Set Segment Transformation, 10–7 to 10–8
Set String Mode, 5–39 to 5–40
Set Stroke Mode, 5–33 to 5–34
Set Text Representation, 8–7 to 8–8
Set Valuator Mode, 5–35 to 5–36
Set Visibility, 10–9 to 10–10
Set Workstation Viewport, 4–16 to 4–17
Set Workstation Window, 4–14 to 4–15
Text, 9–10 to 9–14
Update Workstation, 4–9 to 4–10
Write Item to Metafile, 7–3 to 7–4
Workstation handlers, 1–5
Workstation State List, 2–2
for workstation handlers, 3–25 to 3–31
Workstation transformation, 2–4
Workstation viewport, 2–3
Workstation window, 2–3
WORKSTATION_CATEGORY, 3–5
World coordinates, 2–3
World viewport, 2–3
World window, 2–3
WORLD_VIEWPORT, 3–27
WORLD_WINDOW, 3–27
Write Item to Metafile function, 7–3 to 7–4
Writing handler, 1–4
Writing handlers, 1–2
WSTYPE, 3–26

X

XFORM
in WSL, 3–28, 3–29

Reader's Comments

Building a DEC GKS Workstation Handler System
Order No. AA–MJ34A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Cut Along Dotted Line

Reader's Comments

Building a DEC GKS Workstation Handler System
Order No. AA–MJ34A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Cut Along Dotted Line