DEC GKS Reference Manual
Volume I

Order Number: AA–HW43C–TE

April 1989

This document is an encyclopedic reference to the DEC GKS level 2c run-time functions. This volume contains information on the DEC GKS control, output, output attribute, transformation, input, segment, metafile, and error-handling functions. DEC GKS software users can review release notes by typing HELP GKS RELEASE_NOTES on the DCL command line.

Revision/Update Information:	This revised document supersedes the *VAX GKS Reference Manual Volume I* (Order No. AI–HW43B–TE).
Operating System and Version:	VMS Version 4.7 or higher. ULTRIX Version 3.0 or higher. VAXstation requirement: VAXstation Windowing Software Versions 3.1 or higher, or DECwindows Version 1.0.
Software Version:	DEC GKS Version 4.0

digital equipment corporation
maynard, massachusetts

First Printing March 1984
Revised November 1984, May 1986, 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™

ZK5203

Contents

Examples

Figures

Tables

Preface

Manual Objectives

This manual provides encyclopedic reference to the DEC Graphical Kernel System (GKS) and provides examples illustrating DEC GKS function calls. DEC GKS is a level 2c GKS implementation. For more information concerning GKS implementation levels, refer to Chapter 1, Introduction to DEC GKS.

NOTE

Before reading this manual, you should review the DEC GKS release notes by typing the following:

```
$ HELP  GKS  RELEASE_NOTES RETURN
```

Intended Audience

This manual is intended for experienced application programmers who need to reference information concerning the DEC GKS functions. Readers should be familiar with one high-level language and the DIGITAL Command Language (DCL). (For more information concerning DCL, refer to the *VAX/VMS DCL Dictionary*.)

Refer to the DEC GKS Binding Reference Manuals for information specific to the binding you use with DEC GKS. The available bindings for DEC GKS Version 4.0 are FORTRAN, C, and GKS$. These manuals are designed for the experienced user of DEC GKS who needs to know the binding syntax and brief argument descriptions.

Although there are lengthy introductions at the beginning of each of the chapters, this manual is not tutorial in nature. New users who need tutorial information and moderately experienced users needing programming suggestions should refer to the *DEC GKS User Manual*.

Document Structure

This manual is contained in two volumes. Volume I contains the following information:

- Chapter 1, Introduction to DEC GKS, provides an introduction to the DEC GKS product and to the format of this reference manual.
- Chapter 2, Compiling, Linking, and Running DEC GKS Programs on VMS, provides information about DEC GKS and the VMS operating system.
- Chapter 3, Compiling, Linking, and Running DEC GKS Programs on ULTRIX, provides information about DEC GKS and the ULTRIX operating system.
- Chapter 4, Control Functions, provides information concerning the establishment of the DEC GKS and workstation environments.
- Chapter 5, Output Functions, provides information concerning the generation of output primitives.
- Chapter 6, Output Attribute Functions, provides information concerning the output attributes.
- Chapter 7, Transformation Functions, provides information concerning the normalization and workstation transformations.
- Chapter 8, Input Functions, provides information concerning input.
- Chapter 9, Segment Functions, provides information concerning the storage of output primitives in segments.
- Chapter 10, Metafile Functions, provides information concerning long-term storage of graphical images.
- Chapter 11, Error-Handling Functions, provides information concerning error-handling by the application program.

Volume II of this manual contains the following information:

- Chapter 12, Inquiry Functions, provides information concerning the acquisition of DEC GKS and workstation status information.
- The appendixes, which include the following:
  - Appendix A, DEC GKS Supported Workstations
  - Appendix B, DEC GKS Constants

- Appendix C, DEC GKS Attribute Values
- Appendix D, DEC GKS Error Messages
- Appendix E, DEC GKS Metafile Structure
- Appendix F, Language-Specific Programming Information
- Appendix G, DEC GKS Device-Independent Fonts
- Appendix H, DEC GKS Color Chart
- Appendix I, DEC GKS GDPs and Escapes
- Appendix J, DEC GKS Specific Input Values

Associated Documents

You may find the following documents useful when using DEC GKS:

- *DEC GKS User Manual*—For programmers who need tutorial information or guides to programming technique.
- *DEC GKS FORTRAN Binding Reference Manual*—For programmers who need specific syntax and argument descriptions for the FORTRAN binding.
- *DEC GKS GKS$ Binding Reference Manual*—For programmers who need specific syntax and argument descriptions for the GKS$ binding.
- *DEC GKS C Binding Reference Manual*—For programmers who need specific syntax and argument descriptions for the C binding.
- *DEC GKS Device Specifics Reference Manual*—For programmers who need information about specific devices.
- *Building a DEC GKS Workstation Handler System*—For programmers who need to build DEC GKS workstation graphics handlers.
- *Building a DEC GKS Device Handler System*—For programmers who need to provide support for a device unsupported by the DEC GKS graphics handlers.
- *DEC GKS Installation Guide*—For system managers who install the DEC GKS software, including the Run-Time installation, on VMS and ULTRIX operating systems.

Conventions

Convention	Meaning
RETURN	The symbol RETURN represents a single stroke of the RETURN key on a terminal.
$ RUN GKSPROG RETURN	In interactive examples, the user's response to a prompt is printed in red; system prompts are printed in black.
INTEGER X . . . X = 5	A vertical ellipsis indicates that not all of the text of a program or program output is illustrated. Only relevant material is shown in the example.
option, . . .	A horizontal ellipsis indicates that additional arguments, options, or values can be entered. A comma that precedes the ellipsis indicates that successive items must be separated by commas.
[output-source, . . .]	Square brackets, in function synopses and a few other contexts, indicate that a syntactic element is optional.
deferral mode	All names of the DEC GKS description table and state list entries, and of the workstation description table and state list entries, are italicized.

Summary of Technical Changes

New and Changed Features

This manual is a revision of the *DEC GKS Reference Manual* and contains the following new and changed features.

- All device specific appendixes in Version 3.0, K through R, are now documented in the *DEC GKS Device Specifics Reference Manual.*
- The sections of Appendix B, DEC GKS Constants, in the *DEC GKS Reference Manual,* describing the following:
  - An error handling state description
  - The FORTRAN binding constant name GGFACP
- The sections of Appendix D, DEC GKS Error Messages, in the *DEC GKS Reference Manual,* describing a new message.
- The sections of Appendix F, Language-Specific Programming Information, in the *DEC GKS Reference Manual,* describing corrected type definitions in the Programming in VAX Pascal section.
- The sections of Appendix I, DEC GKS GDPs and Escapes, in the *DEC GKS Reference Manual,* describing the following:
  - The new fill area set GDP
  - New escape functions
  - Corrections to the Set Writing Mode function
- The sections of Appendix J, DEC GKS Specific Input Values, in the *DEC GKS Reference Manual,* describing Locator and Stroke Input classes.

Chapter 1

Introduction to DEC GKS

The Graphical Kernel System (GKS) is a set of graphics functions that can be used by numerous types of graphics applications to produce two-dimensional pictures on graphics output devices. GKS is defined by the ANSI X3.124-1985 and the ISO 7942-1985 standards. DEC GKS adheres to both standards. When this manual refers to the GKS standard, the reference applies to both standards.

The GKS standard provides a functional standard, and syntactical standards called *language bindings*. The functional standard determines the effects produced by a particular GKS function, but does not specify the function name or the number of function parameters. Therefore, a given function in two different GKS implementations can produce the same effects, but may have a different function name or a different number of parameters.

DEC GKS implements the functional standard using function names beginning with the prefix GKS$. These functions should be used when programming with the VMS implementation of DEC GKS. If you use the GKS$ functions, you have to edit your program if you want to transport the program across systems or across GKS implementations.

DEC GKS also implements approved syntactical language bindings. For DEC GKS Version 4.0, these include the GKS FORTRAN and GKS C bindings. The language bindings in general, and specifically the FORTRAN and C bindings, provide standard function names and a standard number of function parameters. If you write programs to be transported across systems or across GKS implementations, you should use the appropriate language binding.

1.1 GKS Function Categories

The DEC GKS function categories are as follows:

- Control
- Output
- Output attribute
- Transformation
- Input
- Segment
- Metafile
- Error-handling
- Inquiry

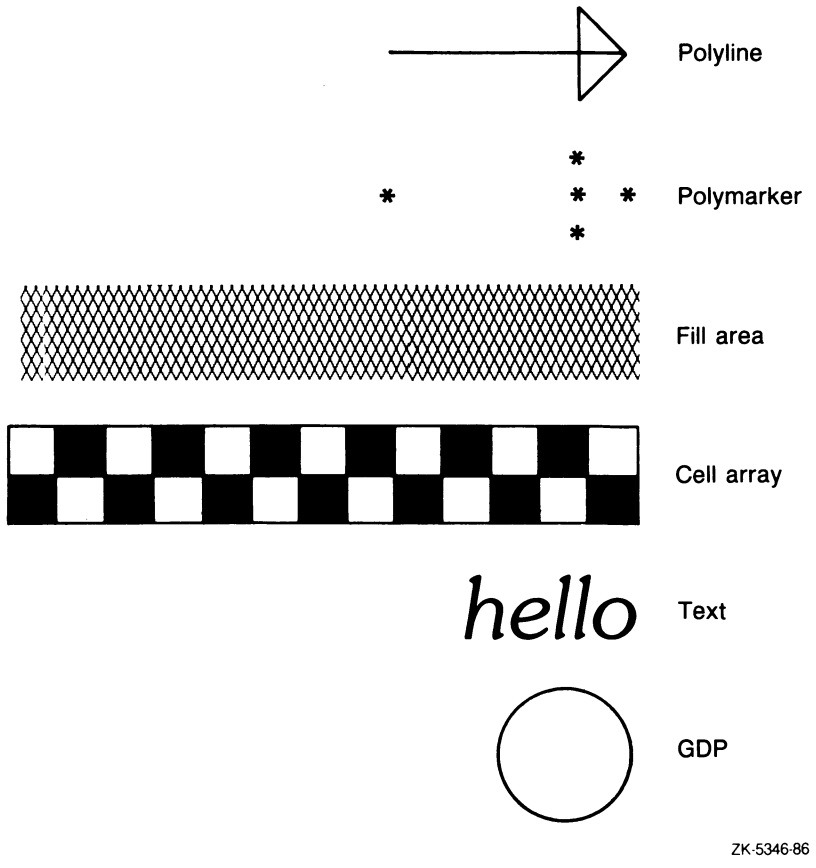
The control functions determine which DEC GKS functions you can call at a given point in your program. They also control the buffering of output and the regeneration of segments on the workstation surface.

The output functions produce picture components, called *primitives*, of the following types:

- Polylines—Lines
- Polymarkers—Symbols
- Fill areas—Filled polygons
- Text—Character strings
- Cell Array—Filled cells of a rectangle
- Generalized drawing primitives—A workstation-dependent image such as a circle

Figure 1–1 illustrates possible representations of output primitives.

Figure 1–1: Possible DEC GKS Primitives



Output attributes affect the appearance of a primitive. For instance, by changing the line type attribute, you can produce solid, dashed, dotted, or dashed-dotted lines.

Transformations affect the composition of the graphical picture and the presentation of that picture. There are *normalization* and *workstation* transformations. The normalization transformations allow you to use various coordinate ranges for different primitives within a single picture. In this way, you can use a coordinate range that suits each particular primitive in a large picture.

The workstation transformations control the portion of the picture that you see on the workstation's surface, and the portion of the surface used to display the picture. Using workstation transformations, you can pan across a picture, zoom in to a picture, or zoom out of a picture.

The input functions allow an application to accept data from a user.

The segment functions store and manipulate groups of primitives called *segments.*

The metafile functions allow you to store and to recall an audit of calls to DEC GKS functions. Using metafiles, you can store a DEC GKS session so that another application can interpret that session, thus reproducing the picture created by the original application. For more information concerning metafiles, refer to Chapter 10, Metafile Functions.

The error-handling functions allow you to invoke a user-written error handler when a call to another DEC GKS function generates an error. For more information concerning error-handling, refer to Chapter 11, Error-Handling Functions.

The inquiry functions obtain either default or current information from the DEC GKS data structures.

If you need more tutorial information concerning DEC GKS concepts, refer to the *DEC GKS User Manual.*

1.2 GKS Levels

The GKS standard defines levels of a GKS implementation that address the most common classes of graphic devices and application needs. The levels are determined primarily by input and output capabilities. The output level values are represented by the characters m, 0, 1, and 2. The input level values are represented by the characters a, b, and c.

The DEC GKS software is a level 2c implementation, incorporating all of the GKS output capabilities (level 2) and all of the input capabilities (level c). This manual uses the term DEC GKS when describing the 2c level DEC GKS product.

Figure 1–2 defines the 12 upwardly compatible levels of GKS. DEC GKS implements all listed functionality.

Figure 1–2: Functionality by GKS Levels

Output Levels	Input Levels a	b	c
m	No input, minimal control, individual attributes, one settable normalization transformation, subset of output and attribute functions.	Request input, set operating mode and initialize functions for input devices, no pick input.	Sample and event input no pick.
0	Basic control, bundled attributes, multiple normalization transformations, all output and attribute functions, optional metafiles.	Set viewport input priority.	All of level mc, above.
1	Full output including settable bundles, multiple workstations, basic segmentation, no workstation independent segment storage, metafiles.	Request pick, set operating mode and initialize functions for pick input.	Sample and event input for pick.
2	Workstation independent segment storage	All of level 1b, above.	

ZK-5027-86

Pick input is one of the DEC GKS logical input classes used to specify segments present on the surface of a device. Request, sample, and event are GKS input operating modes. DEC GKS supports all three input operating modes. For more information on pick input or operating modes, refer to Chapter 8, Input Functions.

Workstation independent segment storage (WISS) provides a way to store segments so that one segment can be transported to different devices. For more information, refer to Chapter 9, Segment Functions.

1.3 Coordinate Range Format

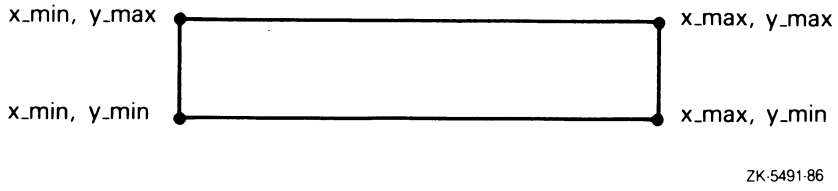
When specifying a coordinate range, whether the range is located in world coordinate space, normalized device coordinate space, or device coordinate space, this manual uses a single notation.

The syntax of this rectangular range specification is as follows:

([x_min, x_max] X [y_min, y_max])

Figure 1–3 illustrates the rectangular coordinate area.

Figure 1–3: Coordinate Range Presentation



For more information concerning the DEC GKS coordinate systems, refer to Chapter 7, Transformation Functions.

1.3.1 Standard Escape/GDP Data Records

When calling the functions GKS$ESCAPE or GKS$GDP (generalized drawing primitive), you may need to pass a data record. DEC GKS has a standard escape/GDP data record that contains up to three integer components and four array addresses.

To use an escape or GDP data record, you need to perform the following tasks:

1. Look up the escape or GDP description in Appendix I, DEC GKS GDPs and Escapes, in the *DEC GKS Reference Manual.*
2. Determine the size and contents of the required data record (if one is required).
3. Declare the data record as determined by your particular programming language. Each of the seven components of the data record is an integer value. The record is read only, passed by reference.
4. Pass to GKS$ESCAPE or GKS$GDP only the data record components required by the escape or GDP. For instance, if an escape or GDP only requires 5 data record components, omit values from components 6 and 7.
5. Pass to GKS$ESCAPE or GKS$GDP the exact size of the valid portion of the data record, as specified in Appendix I, DEC GKS GDPs and Escapes, in the *DEC GKS Reference Manual.* For instance, if an escape or GDP requires 5 valid components to the data record, then pass the value 20 as the data record size (each component being a longword in length).

The DEC GKS standard escape/GDP data record is as follows.

Position	Data Type	Description
1	Integer	Number of integer values passed in the data record.
2	Integer	Number of real values passed in the data record.
3	Integer	Number of string addresses passed in the data record.
4	Integer (address)	Address of array of integers with exactly as many elements as the number specified in component number 1.
5	Integer (address)	Address of array of real numbers with exactly as many elements as the number specified in component number 2.
6	Integer (address)	Address of array of string lengths with exactly as many elements as the number specified in component number 3.
7	Integer (address)	Address of array of string addresses with exactly as many elements as the number specified in component number 3.

After performing a task, some escape functions pass information back to you by use of an output data record. This output data record is identical in format to the input data record, except that the output record's components are modifiable. You pass the buffer sizes in the first three components and the addresses of your buffers in the last four components. DEC GKS modifies the first three components to contain the number of elements DEC GKS actually used to write output data to each of the corresponding buffers.

If you are using an escape function and you need to determine the size required by the entire output data record buffer, you can pass the value 0 to the output record buffer size (documented as the argument record_buffer_length in the GKS$ESCAPE function description, described in Chapter 4, Control Functions, in the *DEC GKS Reference Manual*). When you pass the value 0 as this argument, GKS$ESCAPE does *not* perform the escape, but instead returns the size of the output data record to argument record_size. In this manner, you can be sure that you declared an output data record buffer that is large enough to hold the entire data record.

To place array addresses in the fourth, fifth, sixth, and seventh components of the data record, you need to use a technique specific to your programming language. For instance, using VAX FORTRAN, you can use the %LOC built-in function. For more information concerning addresses and pointers, refer to the documentation set for your programming language. For more information concerning the use of %LOC and data records, refer to the choice input examples in Chapter 8, Input Functions, in the *DEC GKS Reference Manual*.

For more information, refer to Appendix I, DEC GKS GDPs and Escapes, in the *DEC GKS Reference Manual* or to the *DEC GKS Device Specifics Reference Manual.*

NOTE

Remember that the DEC GKS input data records have a format that is completely different from the DEC GKS standard escape/GDP data record format. To review the GKS standard input data records, refer to Chapter 8, Input Functions, in the *DEC GKS Reference Manual.* To review the actual data records required by the DEC GKS graphics handlers, refer to Appendix J, DEC GKS Specific Input Values, in the *DEC GKS Reference Manual.*

1.4 Function Presentation Format

This section describes the format used to provide information about each of the DEC GKS functions that use the GKS$ prefix. If you are using a language binding, you can find a similar discussion concerning the format of binding function descriptions at the beginning of the appropriate language binding book.

The following sections describe the format used to present each of the DEC GKS function descriptions.

1.4.1 Function Description

Each function description in this manual begins with the English version of the function name at the top of the page. This function name is located at the top of each subsequent page of the function description.

The first paragraph of the function description list the following items:

- The GKS standard function name.
- The valid operating states during which a call to the function is permitted (for more information, refer to Chapter 4, Control Functions).

Following the listed information is a short description of the function. Within this description is pertinent information about the DEC GKS operating state, the DEC GKS description table and state list, and the workstation description table and state list.

1.4.2 Function Syntax

The syntax section of the function description lists the syntax of a call to the DEC GKS function. The syntax of each DEC GKS function call is available for the GKS$, FORTRAN, and C bindings. This syntax includes the argument list for each binding.

Following each syntax section is an argument section that lists each GKS$ argument on a separate line.

All of the DEC GKS functions always return a longword condition status value. For a description of the longword status value, refer to Appendix D, DEC GKS Error Messages. For information concerning DEC GKS error handling, refer to Chapter 11, Error-Handling Functions.

1.4.3 Argument Descriptions

The argument descriptions for each of the functions appear as follows:

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is an integer value that identifies an open workstation.

The arguments passed to DEC GKS functions must be of specific data types and they must be passed by specific mechanisms. In the function descriptions, these data types are described in the list following each of the argument names.

For each argument, the listed values include:

- The data type of the argument
- The type of access made by the function
- The argument-passing mechanism and form

Most of the passing mechanisms required by DEC GKS functions are the default mechanisms of VAX FORTRAN. (This manual clearly documents those functions requiring different passing mechanisms in the section labeled Arguments within each function description.) Refer to the *DEC GKS C Binding Reference Manual* for information about C binding passing mechanisms.

The other VAX high-level languages use different default passing mechanisms. If you are using a high-level language other than FORTRAN, you may need to use the argument-passing extensions for that language. The include file for some languages (for example, Pascal, BASIC, and PL/I) define the default passing mechanisms for each DEC GKS function call.

Some of the descriptions of data types in this manual are not worded in exactly the same manner as in the VMS documentation. For instance, when this manual says that an argument is of the data type "real," the corresponding VMS data type is "F_floating point." The following list presents the notation used in this manual and the corresponding VMS notation:

GKS Type/Mechanism	Corresponding VMS Type/Mechanism
Integer	Longword integer (signed)
Real	F_floating point
String	Character-coded text string
Address (record)	Longword integer (signed) This is an address of a data record.
Type: array (integer) Mechanism: by reference	Type: longword integer (signed) Mechanism: by reference, array reference

For a complete discussion of the argument-passing mechanisms, refer to the *VAX/VMS Run-Time Library Routines Reference Manual*. For information concerning language-specific passing extensions, refer to the appropriate VAX high-level language manual.

1.4.4 Error Message List

The function descriptions list all errors that can possibly be generated by a call to that specific DEC GKS function. For a complete description of the error message, the possible cause, and the possible user action, refer to Appendix D, DEC GKS Error Messages.

1.4.5 Program Examples

Each function description either lists a program example or refers you to another example that calls the specified DEC GKS function. All functions are written in FORTRAN for use with the VT241, for consistency in presentation. FORTRAN-specific constructs are flagged. However, if you are unfamiliar with FORTRAN, you may wish to review the following list of FORTRAN-specific constructs used in the program examples in this manual:

Construct	Description
IMPLICIT NONE	This statement prevents the VAX FORTRAN compiler from implicitly declaring variable names that you have not declared.
C	This character, located in the first column of the line, signifies that the entire line contains a comment.
*	This character, located in column six, is a continuation character. This character signifies that the previous line of code continues onto the line marked with the asterisk (*).
DATA	The DATA statement initializes program variables with data.
CHARACTER*80	This identifier is used to declare a character string of length 80.
INTEGER var(3)	This declaration declares a three-element array of type INTEGER.
%DESCR %VAL %REF	These constructs are argument list built-in functions used to pass arguments by descriptor, by value, and by reference.
LEN	This construct is a built-in function that returns the length of a string.
%LOC(array)	This built-in function returns the address of its argument.

In many of the FORTRAN examples in this book, the following lines of code cause the program to pause, so that you can view the image on the workstation surface as it is being created.

```
      .
      .
      .
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( 1, GKS$K_POSTPONE_FLAG )
      READ(5,*)
      .
      .
      .
```

Since DEC GKS allows the VT241 to *defer*, or buffer, output, you have to update the screen with a call to GKS$UPDATE_WS in order to view the picture created by all previous function calls in the program. The FORTRAN READ statement causes the pause in program execution.

Since the rate of deferral may differ on various workstations, you may wish to use the function GKS$INQ_WS_DEF_AND_UPDATE to check the current deferral mode. If the deferral mode is anything other than GKS$K_ASAP, you may wish to update the workstation surface occasionally when you are debugging your program. If you want to change the deferral mode so that the workstation surface is always current, you can call the function GKS$SET_DEFER_STATE to change the current deferral mode.

For detailed information concerning the DEC GKS deferral mode, refer to Chapter 4, Control Functions.

Also, all program examples include the following line:

```
      .
      .
      .
      CALL GKS$OPEN_WS( 1, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      .
      .
      .
```

To convert the program for use with a device other than a VT241, change the constant GKS$K_VT240 to the appropriate workstation constant value (refer to Appendix A, DEC GKS Supported Workstations), and change any device specific information within the program (such as bundled attribute values). The device-specific information within each program is noted as such.

After many of the program examples, there is an illustration representing the graphical image generated on the surface of the VT241. Since there are visual differences between the written page and the workstation surface, the image may appear different on your device surface. Also, different devices produce different results.

For instance, the color may be a different hue or lines may not as perfectly smooth as presented in the figure. The figures in this manual serve the purpose of showing relative positioning, general color (where applicable), and general shape of the graphical image on the surface of the VT241.

1.4.6 Returning a Data Record

The DEC GKS FORTRAN binding does not return data records. This restriction conforms with the GKS Standard. Use the GKS$ function with FORTRAN if you want to return the data record.

Chapter 2

Compiling, Linking, and Running DEC GKS Programs on VMS

The DEC GKS functions that begin with the prefix GKS$ are designed to be used on one of the VMS systems. Those functions meet the *functional* GKS standard. In other words, they perform the necessary tasks as designated by the GKS standard.

However, these functions are in no way meant to meet a *syntactical* standard. For instance, the DEC GKS function GKS$CELL_ARRAY might have a different number of arguments than the cell array function in another GKS implementation. As a result, programs written using the GKS$ interface are not easily transportable; you have to edit the function names, and possibly the number and order of function arguments.

2.1 VMS Programming Considerations

The specific method for using DEC GKS software depends on the features and conventions of each VAX language. This section discusses general issues that must be considered when using any VAX language with DEC GKS.

NOTE

Some of the VAX languages have language-specific requirements for using VAX GKS. For a complete discussion, you should refer to Appendix F, Language-Specific Programming Information, before coding your programs. For a discussion of the capabilities of each of the DEC GKS supported physical devices, refer to the *DEC GKS Device Specifics Reference Manual.*

2.1.1 Online Help

DEC GKS provides an online HELP library. To access this information, type the following:

```
$ HELP GKS RETURN
```

Before using the DEC GKS software, you should review the release notes for information pertinent to the current release. To review the release notes, type the following:

```
$ HELP GKS RELEASE_NOTES RETURN
```

2.1.2 Capabilities of Supported Devices

In many applications, you may wish to write completely device-independent programs. In this way, you can run your programs using different devices without having to rewrite your programs. The *DEC GKS User Manual* outlines the procedure for device-independent programming using DEC GKS.

However, you may wish to review the range of capabilities of the DEC GKS supported devices, or you may wish to write device-dependent subroutines within your application. In any instance, it is helpful to review the device-specific appendixes in this manual before you begin coding your application. The device-dependent appendixes contain information concerning predefined bundle index representations, color capabilities, initial input values, bit masks as workstation type values, supported escape functions for that particular device, and similar information.

2.1.3 Calling Sequences

Each DEC GKS function requires a specific calling sequence. The calling sequence indicates the elements included in the language statement that calls the function, and the order of those elements. The three elements are the following:

- **Call Type**
  High-level VAX languages call DEC GKS functions with CALL statements or function references. For example, when using FORTRAN, you can use a CALL statement to call DEC GKS functions.

- **Function Identifier**

  All DEC GKS function names begin with the prefix GKS$. FORTRAN binding names begin with an uppercase G, and C binding names begin with a lowercase g. The remainder of the name indicates the operation performed by the function.

  If writing programs to be transported across systems or across GKS implementations, use the appropriate language binding functions. Refer to the *DEC GKS FORTRAN Binding Reference Manual* and the *DEC GKS C Binding Reference Manual* for information concerning the FORTRAN and C binding function names.

- **Argument List**

  Arguments that are passed to DEC GKS functions must be listed in the order shown in the syntax descriptions contained in this manual. See Section 3.1.4.1 for more information concerning the function description format used in this manual. The various language binding functions may have an argument list that is different from the corresponding GKS$ function.

The specific requirements for writing calls and passing arguments to DEC GKS functions vary from one language to another. Whatever the language of the calling program, DEC GKS functions expect the following:

- Integer arguments to be 32-bit longwords passed by reference.
- Real numbers to be in single-precision, floating-point format passed by reference.
- Character strings to be passed by string descriptors.
- Arrays to be passed either by reference or by descriptor, depending on the particular DEC GKS function.

Each language may have specific requirements concerning the language-specific calling sequence. For a discussion of language-specific programming concerns, refer to Appendix F, Language-Specific Programming Information.

NOTE

For all languages that need to declare DEC GKS functions as external functions, you should type the appropriate language definition file to determine the actual function parameter identifiers specified in the DEC GKS code. See Section 1.3 for more information concerning the language definition files.

2.1.4 Constants and Include Files

DEC GKS constants are symbolic names that are syntactically equivalent to literal integer constants. These constants are used in the following ways:

- As arguments to DEC GKS functions.
- As literal values to which you can compare a returned value from an inquiry function (for example, you can compare the return value, from a call to the function GKS$INQ_WS_TYPE, to the constant GKS$K_VT125).
- As literal completion status codes to which you can compare a function return value.

Many DEC GKS functions use constants as arguments, as shown by the following function call:

```
GKS$CLEAR_WS( 1, GKS$K_CLEAR_ALWAYS )
```

You can compare one of the completion status codes to a function return value, as follows:

```
   .
   .
   .
IF ( GKS$_SUCCESS = GKS$ACTIVATE_WS( 1 ) )
   .
   .
   .
```

Most DEC GKS constants begin with the prefix GKS$K and are defined in a definition file. All DEC GKS completion status code constants begin with the prefix GKS$_ERROR_ or DECGKS$_ERROR_NEG_ and are defined in a separate definition file. All DEC GKS bit mask constants begin with the prefix GKS$M_.

You can either specify a literal value as an argument to a DEC GKS function, or you can include the language definition files and use a defined constant name instead. The use of constants adds to program legibility and program documentation.

To review the list of DEC GKS constants, refer to Appendix B, DEC GKS Constants. To review the list of DEC GKS completion status code constants, refer to Appendix D, DEC GKS Error Messages.

2.1.4.1 Including Definition Files

You use DEC GKS software primarily by placing calls to DEC GKS functions in your program. However, when using DEC GKS, you need statements in your program other than calls to GKS functions. The specific statements that are needed depend on the VAX language you use. (For more information, refer to Appendix F, Language-Specific Programming Information).

DEC GKS constants and their values must be made available to all programs using DEC GKS regardless of the VAX language you use. All VAX high-level languages that use DEC GKS have a method for inserting an external file into the program source code stream at compile time. Incorporating an external file is the method for making DEC GKS constants available.

Your installation kit has been supplied with several files that contain DEC GKS constants and separate files that contain DEC GKS completion status code constants. You incorporate these files into your program with a statement that is appropriate to the language you are using.

For example, BASIC provides the %INCLUDE statement for inserting an external file into a program. Therefore, any BASIC program that uses DEC GKS should contain the following statement:

```
%INCLUDE "SYS$LIBRARY:GKSDEFS.BAS"
```

In the previous statement, the identifier SYS$LIBRARY is the logical name of the directory that contains the files containing DEC GKS constants.

The language definition files located in SYS$LIBRARY are as follows:

- GKSDEFS.ADA for VAX™ Ada®
- GKSDEFS.BAS for VAX BASIC
- GKSDEFS.R32 for VAX BLISS
- GKSDEFS.H for VAX C
- GKSDEFS.LIB for VAX COBOL
- GKSDEFS.FOR for VAX FORTRAN using the GKS$ functions
- GKSDEFS.BND for VAX FORTRAN using the FORTRAN binding functions
- GKSDEFS.PAS for VAX Pascal
- GKSDEFS.PLI for VAX PL/I routines declared as procedures (no value returns)
- GKSDEFS.PL2 for VAX PL/I routines declared as functions

™ VAX is a trademark of Digital Equipment Coorporation.

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

The completion status code definition files located in SYS$LIBRARY are as follows:

- GKSMSGS.ADA for VAX Ada
- GKSMSGS.BAS for VAX BASIC
- GKSMSGS.R32 for VAX BLISS
- GKSMSGS.H for VAX C
- GKSMSGS.LIB for VAX COBOL
- GKSMSGS.FOR for VAX FORTRAN
- GKSMSGS.PAS for VAX Pascal
- GKSMSGS.PLI for VAX PL/I

Each file includes comments that describe the exact method for using a given definition file.

2.1.5 Compiling, Linking, and Running Your Programs

A program that uses DEC GKS function calls should be compiled and executed as any other program. Use the compile command that is appropriate to the language you are using and use the RUN command to execute the program image.

DEC GKS functions are supplied as an installed shareable image library. An installed shareable image makes linking faster and easier. Also, using DEC GKS as a shareable image makes your program's resulting .EXE file smaller.

The symbols in the DEC GKS image have been inserted in the system image library. Therefore, to link a compiled program to DEC GKS, you only need to specify the name of your program's object file on the command line, as follows:

```
$ LINK  MYPROG.OBJ RETURN
```

However, if you are using language binding functions in your program, you need to link your program's object file with the appropriate binding object library. To link your program to the FORTRAN binding object library, issue the following command:

```
$ LINK  MYPROG.OBJ, SYS$LIBRARY:GKSFORBND/LIBRARY RETURN
```

2.1.6 Logical Names and DEC GKS Programming

NOTE

If you are unfamiliar with VMS logical names, then you may wish to review the *Introduction to VAX/VMS* before reading this section.

In many DEC GKS programs, the execution of your application appears as follows:

```
❶       CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
❷       CALL GKS$OPEN_WS( 1, GKS$K_CONID_DEFAULT,
      * GKS$K_WSTYPE_DEFAULT )
        CALL GKS$ACTIVATE_WS( 1 )
           .
           .
           .
C       Release the DEC GKS and workstation environments.
        CALL GKS$DEACTIVATE_WS( 1 )
        CALL GKS$CLOSE_WS( 1 )
        CALL GKS$CLOSE_GKS()
```

The following numbers correspond to the numbers in the previous example:

❶ In this call to GKS$OPEN_GKS, the logical name SYS$ERROR is the only argument to the function. This argument tells DEC GKS where to write generated error messages.

If you pass the logical name SYS$ERROR (or the value 0), DEC GKS translates this logical name and writes the error messages to the location specified by the translation. By default, SYS$ERROR translates to the logical name TT, which in turn translates to your process's default device connection (error messages appear on your terminal's display surface).

If you choose, you can specify a VMS file specification as an argument to GKS$OPEN_GKS. In this way, you have a permanent record of generated error messages for use during program debugging.

❷ The constant GKS$K_CONID_DEFAULT (or the value 0) tells DEC GKS to translate the logical name GKS$CONID in order to determine the name of the device connection.

The constant GKS$K_WSTYPE_DEFAULT (or the value 0) tells DEC GKS to translate the logical name GKS$WSTYPE in order to determine the name of the workstation type.

Consequently, you can use the DEFINE or ASSIGN command on the DCL command line to define the logical names to be the connection and type with which you are working, as follows:

```
$ language_compile_command  PROGRAM [RETURN]
$ LINK  PROGRAM [RETURN]
$ DEFINE  GKS$CONID   ttb0 [RETURN]
$ DEFINE  GKS$WSTYPE  13   ! VT241 Color [RETURN]
$ RUN   PROGRAM [RETURN]
    .
    .
    .
$ DEFINE  GKS$CONID   tta0 [RETURN]
$ DEFINE  GKS$WSTYPE  12   ! VT125 Black and White [RETURN]
$ RUN   PROGRAM [RETURN]
    .
    .
    .
```

Before you attempt to define GKS$CONID, you need to perform the following tasks:

1. Make sure that you have allocated the device you need to access. The DCL command SHOW DEVICE provides a list of devices on your system node.
2. Allocate the terminal using the command ALLOCATE (you may need special privileges to allocate the device).
3. Use the command SHOW TERMINAL to make sure that the device's baud rate, parity, and other settings match the settings of the physical device.
4. Define the logical GKS$CONID to be the logical name of the appropriate device connection.

For more information concerning the terminal allocation process, refer to the appropriate commands in the *VAX/VMS DCL Dictionary*.

There may be times when you do not wish to define the DEC GKS logical names. In this case, or if you define an invalid value, DEC GKS translates several logical names in the following order:

1. If the logical name GKS$CONID is undefined, DEC GKS translates the logical name TT.
2. DEC GKS then translates TT, which always defaults to your process's default device connection.

If the logical name GKS$WSTYPE is undefined, then DEC GKS sets the device type to be GKS$K_VT240BW (the value 14, a black and white VT240).

The ability to define GKS$CONID and GKS$WSTYPE provides device independency. For more information concerning device-independent DEC GKS programs, refer to the *DEC GKS User Manual*.

2.1.6.1 Specifying Bit Masks as Workstation Type Values

You have the option of specifying the workstation type value in either a hexadecimal, octal, or decimal longword value. In most cases, it is sufficient to specify the type value in decimal.

However, some of the DEC GKS supported devices allow you to pass a *bit mask* in the first word of the longword workstation type value. For example, the following workstation type specifies default values for the DIGITAL LVP16 plotter:

```
$ DEFINE  GKS$WSTYPE   51 RETURN
```

The following hexadecimal workstation type specifies to DEC GKS to use the LVP16 plotter in landscape mode, with a paper size of 11 x 17 inches:

```
$ DEFINE  GKS$WSTYPE   %x00020033 RETURN
```

For a complete list of all of the available bit masks for a given device, refer to the *DEC GKS Device Specifics Reference Manual.*

Chapter 3

Compiling, Linking, and Running DEC GKS Programs on ULTRIX

The DEC GKS functions that begin with the prefix GKS$ are designed to be used on a DIGITAL system. Those functions meet the *functional* GKS standard. In other words, they perform the necessary tasks as designated by the GKS standard.

However, these functions are in no way meant to meet a *syntactical* standard. For instance, the DEC GKS function GKS$CELL_ARRAY might have a different number of arguments than the cell array function in another GKS implementation. As a result, programs written using the GKS$ interface are not easily transportable; you have to edit the function names, and possibly the number and order of function arguments.

Use the FORTRAN binding, and approved ISO and ANSI standard, for transportability.

3.1 ULTRIX Programming Considerations

The specific method for using DEC GKS software depends on the features and conventions of each VAX language. This section discusses general issues that must be considered when using any VAX language with DEC GKS. For a discussion of the capabilities of each of the DEC GKS supported physical devices, refer to the appropriate device-specific chapter in the *DEC GKS Device Specifics Reference Manual.*

3.1.1 Supported Languages

DEC GKS supports the following languages:

- VAX FORTRAN
- VAX C
- CC (Portable C)

3.1.2 Capabilities of Supported Devices

In many applications, you may wish to write completely device-independent programs. In this way, you can run your programs using different devices without having to rewrite your programs. The *DEC GKS User Manual* outlines the procedure for device-independent programming using DEC GKS.

However, you may wish to review the range of capabilities of the DEC GKS supported devices, or you may wish to write device-dependent subroutines within your application. In any instance, it is helpful to review the *DEC GKS Device Specifics Reference Manual* before you begin coding your application. The device-dependent appendixes contain information concerning predefined bundle index representations, color capabilities, initial input values, bit masks as workstation type values, supported escape functions for that particular device, and similar information.

3.1.3 Calling Sequences

Each DEC GKS function requires a specific calling sequence. The calling sequence indicates the elements included in the language statement that calls the function, and the order of those elements. The three elements are the following:

- **Call Type**
  High-level VAX languages call DEC GKS functions with CALL statements or function references. For example, when using FORTRAN, you can use a CALL statement to call DEC GKS functions.
- **Function Identifier**
  All DEC GKS function names begin with the prefix GKS$. FORTRAN binding names begin with an uppercase G, and C binding names begin with a lowercase g. The remainder of the name indicates the operation performed by the function.

If writing programs to be transported across systems or across GKS implementations, you should use the appropriate language binding functions. Refer to the *DEC GKS FORTRAN Binding Reference Manual* and the *DEC GKS C Binding Reference Manual* for information concerning the FORTRAN and C binding function names.

- **Argument List**

  Arguments that are passed to DEC GKS functions must be listed in the order shown in the syntax descriptions contained in this manual. See Section 3.1.4.1 for more information concerning the function description format used in this manual. The various language binding functions may have an argument list that is different from the corresponding GKS$ function.

The specific requirements for writing calls and passing arguments to DEC GKS functions vary from one language to another. Whatever the language of the calling program, DEC GKS$ binding functions expect the following:

- Integer arguments to be 32-bit longwords passed by reference.
- Real numbers to be in single-precision, floating-point format passed by reference.
- Character strings to be passed by string descriptors.
- Arrays to be passed either by reference or by descriptor, depending on the particular DEC GKS function.

Each language may have specific requirements concerning the language-specific calling sequence. In VAX C, for example, strings are passed by a null terminator. For a discussion of language-specific programming concerns, refer to Appendix F, Language-Specific Programming Information, in the *DEC GKS Reference Manual.*

NOTE

For all languages that need to declare DEC GKS functions as external functions you should type the appropriate language definition file to determine the actual function parameter identifiers specified in the DEC GKS code. See Section 1.3 for more information concerning the language definition files.

3.1.4 Constants and Include Files

DEC GKS constants are symbolic names that are syntactically equivalent to literal integer constants. These constants are used in the following ways:

- As arguments to DEC GKS functions.
- As literal values to which you can compare a returned value from an inquiry function (for example, you can compare the return value, from a call to the function GKS$INQ_WS_TYPE, to the constant GKS$K_VT125).
- As literal completion status codes to which you can compare a function return value.

NOTE

Constants (defines) for the bindings are in the binding specific include files.

Many DEC GKS functions use constants as arguments, as shown by the following function call:

```
GKS$CLEAR_WS( 1, GKS$K_CLEAR_ALWAYS )
```

You can compare one of the completion status codes to a function return value, as follows, in this C example:

```
    .
    .
    .
if ( GKS$_SUCCESS == GKS$ACTIVATE_WS( 1 ) )
    .
    .
    .
```

Most DEC GKS constants begin with the prefix GKS$K_ and are defined in a definition file. All DEC GKS completion status code constants begin with the prefix GKS$_ERROR_ or DECGKS$_ERROR_NEG_ and are defined in a separate definition file. All DEC GKS bit mask constants begin with the prefix GKS$M_.

You can either specify a literal value as an argument to a DEC GKS function, or you can include the language definition files and use a defined constant name instead. The use of constants adds to program legibility and program documentation.

To review the list of DEC GKS constants, refer to Appendix B, DEC GKS Constants, in the *DEC GKS Reference Manual*. To review the list of DEC GKS completion status code constants, refer to Appendix D, DEC GKS Error Messages, in the *DEC GKS Reference Manual*.

3.1.4.1 Including Definition Files

You use DEC GKS software primarily by placing calls to DEC GKS functions in your program. However, when using DEC GKS, you need statements in your program other than calls to GKS functions. The specific statements that are needed depend on the VAX language you use. (For more information, refer to Appendix F, Language-Specific Programming Information, in the *DEC GKS Reference Manual.*)

DEC GKS constants and their values must be made available to all programs using DEC GKS regardless of the language you use. All high-level languages that use DEC GKS have a method for inserting an external file into the program source code stream at compile time. Incorporating an external file is the method for making DEC GKS constants available.

Your installation kit has been supplied with files that contain DEC GKS constants and separate files that contain DEC GKS completion status code constants. You incorporate these files into your program with a statement that is appropriate to the language you are using.

For example, the C programming language provides the #INCLUDE statement for inserting an external file into a program. Therefore, any C program that uses the C binding should contain the following statement:

```
#INCLUDE <GKS/gks.h>
```

Any FORTRAN program that uses the FORTRAN binding functions should contain the following statement:

```
INCLUDE '/usr/include/GKS/gksdefs.bnd'
```

The language definition files located in /usr/include/GKS are as follows:

- gksdefs.h for VAX C and CC (GKS$ binding)
- gks.h for VAX C and CC (C binding)
- gksdefs.bnd for VAX FORTRAN using the FORTRAN binding functions

The completion status code definition files located in usr/include/GKS are as follows:

- gksmsgs.h for VAX C

Each file includes comments that describe the exact method for using a given definition file.

3.1.5 Compiling, Linking, and Running Your Programs

A program that uses DEC GKS function calls should be compiled and executed as any other program. Use the compile command that is appropriate to the language you are using. To run an executable program, type the executable file name that you specified.

NOTE

The \RETURN convention indicates that you type the backslash character \, press Return, and then type text on the next line of the screen.

3.1.5.1 Compiling and Linking GKS$ Programs

To compile and link a DEC GKS GKS$ program, use the following syntax:

```
vcc -o application application.c\ RETURN
   -lGKS -lddif -dwt -lcursesX -lc -lX11 -lm -lc RETURN
```

3.1.5.2 Compiling and Linking C Binding Programs

To compile and link a DEC GKS C binding program, use the following syntax:

```
vcc -o application application.c\ RETURN
   -lGKS -lddif -dwt -lcursesX -lc -lX11 -lm -lc RETURN
```

3.1.5.3 Compiling and Linking FORTRAN Binding Programs

To compile and link a DEC GKS FORTRAN binding program, use the following syntax:

```
fort -o application application.for\ RETURN
   -lGKSFORBND -lddif -dwt -lcursesX -lc -lX11 -lm -lc RETURN
```

3.1.6 Environment Variables and DEC GKS Programming

In many DEC GKS programs, the execution of your application appears as follows:

```
❶       CALL GKS$OPEN_GKS( stderr )
❷       CALL GKS$OPEN_WS( 1, GKS$K_CONID_DEFAULT,
      * GKS$K_WSTYPE_DEFAULT )
        CALL GKS$ACTIVATE_WS( 1 )
           .
           .
           .
C       Release the DEC GKS and workstation environments.
        CALL GKS$DEACTIVATE_WS( 1 )
        CALL GKS$CLOSE_WS( 1 )
        CALL GKS$CLOSE_GKS()
```

The following numbers correspond to the numbers in the previous example:

❶ In this call to GKS$OPEN_GKS, the name stderr is the only argument to the function. This argument tells DEC GKS where to write generated error messages.

If you pass the name stderr (or the value 0), DEC GKS writes the error messages to the specified location. By default, stderr goes to the device /dev/tty, which translates to your process's default device connection (error messages appear on your terminal's display surface).

If you choose, you can specify a path name as an argument to GKS$OPEN_GKS. In this way, you have a permanent record of generated error messages for use during program debugging.

❷ The constant GKS$K_CONID_DEFAULT (or the value 0) tells DEC GKS to evaluate the environment variable GKSconid in order to determine the name of the device connection.

The constant GKS$K_WSTYPE_DEFAULT (or the value 0) tells DEC GKS to evaluate the environment variable GKSwstype in order to determine the name of the workstation type.

Consequently, you can use the setenv command to your shell to define the environment variables to be the connection and type with which you are working, as follows:

```
csh> setenv  GKSconid  /dev/tty RETURN
csh> setenv  GKSwstype  13
csh> # VT241 Color RETURN
csh> application RETURN
```

```
csh> setenv  GKSconid  /dev/tt00 RETURN
csh> setenv  GKSwstype  12
csh> # VT125 Black and White RETURN
csh> application RETURN
```

.
.
.

There may be times when you do not wish to define the DEC GKS environment variables. In this case, or if you define an invalid value, DEC GKS translates several environment variables in the following order:

1. If the environment variable GKSconid is undefined, DEC GKS uses logical name /dev/tty for output.
2. If the environment variable GKSwstype is undefined, then DEC GKS sets the device type to be GKS$K_VT240BW (the value 14, a black and white VT240).

The ability to define GKSconid and GKSwstype provides device independency. For more information concerning device-independent DEC GKS programs, refer to the *DEC GKS User Manual.*

3.1.6.1 Specifying Bit Masks as Workstation Type Values

You have the option of specifying the workstation type value in either a hexadecimal, octal, or decimal longword value. In most cases, it is sufficient to specify the type value in decimal.

However, some of the DEC GKS supported devices allow you to pass a *bit mask* in the first word of the longword workstation type value. For example, the following workstation type specifies default values for the DIGITAL LVP16 plotter:

```
csh> setenv  GKSwstype   51 RETURN
```

The following decimal workstation type specifies to DEC GKS to use the LVP16 plotter in landscape mode, with a paper size of 11 x 17 inches:

```
csh> setenv  GKSwstype   %x131123 RETURN
```

For a complete list of all of the available bit masks for a given device, refer to the *DEC GKS Device Specifics Reference Manual.*

Chapter 4

Control Functions

The control functions establish the DEC GKS and workstation environments, and control the workstation surface in a variety of ways. The following list presents the control functions by category:

Category	GKS Functions
GKS Environment	GKS$OPEN_GKS, GKS$CLOSE_GKS
Workstation Environment	GKS$OPEN_WS, GKS$ACTIVATE_WS, GKS$DEACTIVATE_WS, GKS$CLOSE_WS
Display Surface Control	GKS$CLEAR_WS, GKS$REDRAW_SEG_ON_WS, GKSSET_DEFER_STATE, GKSUPDATE_WS
Additional Control	GKS$ESCAPE, GKS$MESSAGE

In a typical program, you need very few lines of code to tell DEC GKS about the type of implementation you are using, the type of device you are using for input or output, and the functionality allowed with that particular type of device. (Input, output, and other types of devices are called *workstations*.) You begin and end most DEC GKS sessions with lines of code similar to the following:

```
C     Establish the DEC GKS environment; write errors to the device
C     represented by the logical name SYS$ERROR.

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )

C     Open the default workstation, on the default device, and give
C     the workstation an identification number. If you are working with
C     a device which supports input, you can request input after this
C     function call, but you cannot generate output.

      CALL GKS$OPEN_WS( 1, GKS$K_CONID_DEFAULT,
     * GKS$K_WSTYPE_DEFAULT )

C     Activate the workstation using its identification number. If the
C     workstation supports output, you can generate output "primitives"
C     after this function call.
```

```
      CALL GKS$ACTIVATE_WS( 1 )
         .
         .
         .
C     Release the DEC GKS and workstation environments.
      CALL GKS$DEACTIVATE_WS( 1 )
      CALL GKS$CLOSE_WS( 1 )
      CALL GKS$CLOSE_GKS()
```

The previous code example initiates actions by the DEC GKS kernel that involve various operating states, tables, and lists. The tables and lists that are accessible at a given time during program execution determine what types of tasks you can perform (tasks such as input requests and output generation). The following sections discuss the DEC GKS kernel, the DEC GKS operating states, and the various tables and lists involved in working with DEC GKS.

4.1 The Kernel, Graphics Handlers, and Description Tables

The DEC GKS *environment* consists of the kernel, one or more graphics handlers, at least two description tables, and a series of state lists. This section discusses all but the state lists, which are described in detail in Section 4.1.2.

The DEC GKS *kernel* performs basic operations that do not depend on capabilities specific to input, to output, or to the use of storage devices. The kernel gives the DEC GKS functions access to the information and tools necessary to perform properly. The kernel operations include calling certain inquiry functions, maintaining certain tables, and issuing calls to graphics handlers.

The DEC GKS *graphics handlers* consist of functions that the kernel calls to perform graphics operations on a particular workstation. The functions include obtaining input, relaying output, and responding to inquiries for workstation-specific information.

DEC GKS supplies graphics handlers for various devices such as the DIGITAL VAXstation II/GPX and the VT240. If you are certain which devices your DEC GKS programs will use, you should review the *DEC GKS Device Specifics Reference Manual*. In this way, you can become familiar with the range of capabilities of a particular device, and you can gain a sense of how the supported devices vary.

The DEC GKS *description table* contains constant information about the GKS implementation you are using. No matter what functions you call in your program or no matter what application you run, the information in the DEC GKS description table does not change. The DEC GKS kernel uses this constant information about DEC GKS to initialize sections of the DEC GKS state list, which is described in Section 4.1.2.

The DEC GKS description table contains information such as the level of GKS you are using (with DEC GKS, level 2c), the number of available workstation types, the list of workstation types, the maximum allowable open workstations, and so forth. The DEC GKS description table is contained in the DEC GKS kernel.

A *workstation description table* contains constant information about one particular device. No matter what functions you call in your program or no matter what application you run, the information in a device's workstation description table does not change, as long as you always use the same graphics handler. Each graphics handler contains a workstation description table describing that particular device. The workstation description table is used to initialize sections of the workstation state list, which is described in Section 4.1.2.

The workstation description table contains information such as the workstation type, the workstation category, the device-specific maximum coordinate values, the default bundled output attribute values, and so forth.

4.1.1 Workstations

A *workstation* provides a common interface through which a DEC GKS application program controls a graphics device. A workstation is usually a physical device that has input and/or output capabilities. (The GKSK_WSCAT_MO, GKSK_WSCAT_MI, and GKS$K_WSCAT_WISS workstations are exceptions and are described in Table 4–1.)

The various capabilities of the workstation determine the *workstation category*. Every workstation description table has an entry for the workstation category of that particular type of workstation. The six workstation categories are as follows:

Table 4–1: Workstation Categories

Category	Description
GKS$K_WSCAT_OUTPUT	A workstation of the category GKS$K_WSCAT_OUTPUT can only display graphical images on a single display surface. A GKS$K_WSCAT_OUTPUT workstation can process all output functions with the possible exception of the device-dependent generalized drawing primitive (GDP) functions. For more information concerning GDPs, refer to Chapter 5, Output Functions.

Table 4–1 (Cont.): Workstation Categories

Category	Description
GKS$K_WSCAT_INPUT	A workstation of the category GKS$K_WSCAT_INPUT can only accept input, which must be accepted by at least one type of logical input device. A GKS$K_WSCAT_INPUT workstation cannot accept the generation of graphical images by DEC GKS output functions. For more information concerning input, refer to Chapter 8, Input Functions.
GKS$K_WSCAT_OUTIN	A workstation of the category GKS$K_WSCAT_OUTIN combines the capabilities of GKS$K_WSCAT_OUTPUT and GKS$K_WSCAT_INPUT workstations. This type of workstation can display graphic images on the workstation surface as well as accept input from the logical input devices. Also, this type of workstation must include at least one logical input device of each class. For more information concerning logical input devices, refer to Chapter 8, Input Functions.
GKS$K_WSCAT_MO	A workstation of the category GKS$K_WSCAT_MO (Metafile Output) stores image-specific data in a file for use in reproducing the graphical image at a later time, perhaps in another application program. For more information concerning metafiles, refer to Chapter 10, Metafile Functions.
GKS$K_WSCAT_MI	A workstation of the category GKS$K_WSCAT_MI (Metafile Input) allows an application program to read and *interpret* items in a file that contains image-specific data used to reproduce a graphic image. The file containing the data to be interpreted must be produced by a GKS$K_WSCAT_MO workstation. For more information concerning metafiles, refer to Chapter 10, Metafile Functions.
GKS$K_WSCAT_WISS	A workstation of the category GKS$K_WSCAT_WISS (workstation independent segment storage) can store output *primitives* as a single unit during the execution of a single application. The group of output primitives is called a *segment*. You can manipulate the group of output primitives within the defined segment as a single entity. The only way to transfer segments from one workstation to another is to store the segment in workstation independent segment storage (WISS) and then copy that segment to whichever open or active workstation you desire. For more information concerning segments, refer to Chapter 9, Segment Functions.

4.1.2 Operating States and State Lists

The previous sections described the constructs, data structures, and tables needed to maintain the static attributes of the DEC GKS implementation and each workstation.

The DEC GKS and workstation states are not static. You can generate many types of output with many different effects on the surface of the workstation, you can use several devices, or you can create different segments. DEC GKS must keep track of the current state of both the DEC GKS and the workstation environments.

For example, the DEC GKS kernel must have access to a flag that designates whether the DEC GKS software has been initialized, allowing access to description tables and other structures. As another example, if you want to output to a workstation, DEC GKS must have access to another flag that designates whether that workstation is active or not.

To keep track of the information that is available to DEC GKS at a given time, DEC GKS maintains its *operating state* and several different *state lists*.

The DEC GKS operating states are as follows:

- GKS$K_GKCL—GKS is closed.
- GKS$K_GKOP—GKS is open.
- GKS$K_WSOP—At least one workstation is open.
- GKS$K_WSAC—At least one workstation is active.
- GKS$K_SGOP—A segment is open.

For a better understanding, review the following code example. (It is similar to the example presented at the beginning of the chapter.) Following the example, Figure 4–1 shows the GKS operating states, the description tables and state lists, and the control functions used to change operating states. You can use the numbers in the example, in the figure, and in the description list to match the lines of code with their effects on the DEC GKS operating state.

```
❶     CALL GKS$OPEN_GKS( 'SYS$ERROR' )
❷     CALL GKS$OPEN_WS( 1, GKS$K_CONID_DEFAULT,
     * GKS$K_WSTYPE_DEFAULT )
❸     CALL GKS$ACTIVATE_WS( 1 )
          .
          .
          .
❹     CALL GKS$CREATE_SEG( 1 )
          .
          .
          .
❺     CALL GKS$CLOSE_SEG()
          .
          .
          .
❻     CALL GKS$DEACTIVATE_WS( 1 )
❼     CALL GKS$CLOSE_WS( 1 )
❽     CALL GKS$CLOSE_GKS()
```

The following numbers correspond to the numbers in the previous example and to the numbers in Figure 4–1:

❶ Before you invoke DEC GKS, the operating state value is GKS$K_GKCL. When DEC GKS is closed, you can call GKS$INQ_OPERATING_STATE, which returns the current operating state, you can call GKS$OPEN_GKS, or you can call DEC GKS functions to log and handle errors. To log and handle errors, DEC GKS maintains the *error state list*. The error state list contains entries that specify the error state and the error log file. If you attempt to call DEC GKS functions while DEC GKS is closed (other than those discussed in this paragraph), the call generates an error message. For more information, refer to Chapter 12, Inquiry Functions, and to Chapter 11, Error-Handling Functions.

In order to perform more tasks using DEC GKS, you must set the operating state to GKS$K_GKOP. To do this, make a call to the control function GKS$OPEN_GKS, and pass to the function the name of an error log file so that DEC GKS knows where to write error messages. If you specify SYS$ERROR, and if you have not redefined that logical name, DEC GKS writes error messages to your terminal.

Once you open DEC GKS, you have enabled access to the DEC GKS description table and the workstation description tables of the supported graphics handlers. By calling GKS$OPEN_GKS, you have also allowed access to the DEC GKS *state list*. The DEC GKS state list contains entries that designate information such as the set of open workstations (if any), the current normalization number, the current character height, and so forth.

Once DEC GKS is open, you can then specify output attributes (refer to Chapter 6, Output Attribute Functions), set normalization transformations (refer to Chapter 7, Transformation Functions), obtain values from the DEC GKS state list, and obtain values from the DEC GKS and workstation

description tables (refer to Chapter 12, Inquiry Functions). If you attempt to call other functions, DEC GKS generates an error message.

❷ To perform further tasks using DEC GKS (such as requesting input), you must open at least one workstation. When you open the first workstation, the DEC GKS operating state changes from GKS$K_GKOP to GKS$K_WSOP (at least one workstation open). To accomplish this, call GKS$OPEN_WS and pass a numeric *workstation identifier*, a physical device name or connection identifier (such as TT, the default connection to your terminal), and a workstation type. (See GKS$OPEN_WS in this chapter for more information.) The workstation identifier is an integer value chosen by you for use in all references in the program to a specific, open or active workstation.

For each workstation you open, there exists a *workstation state list*. This list contains entries that specify whether output is deferred (buffered or on hold), whether you have to update the workstation surface (redraw the picture to fulfill a request for a picture change), whether the workstation surface is empty by DEC GKS definition, whether the picture on the surface represents all of the requests for output made thus far by the application program, and so forth. Many control functions affect the values in this table. See Section 4.2.1 for more information.

Once at least one workstation is open, you can call all functions *except* those functions that open or close DEC GKS, perform output to a workstation, create or insert segments, or write an item to a metafile output (GKS$K_WSCAT_MO) workstation (using the function GKS$WRITE_ITEM). If you attempt to call these functions, DEC GKS produces an appropriate error message.

❸ To perform output on a given workstation, you need to activate that workstation. When you activate the first workstation, the DEC GKS operating state changes from GKS$K_WSOP to GKS$K_WSAC (at least one workstation active). To activate a workstation, call the control function GKS$ACTIVATE_WS, and pass a workstation identifier specifying an open workstation. When DEC GKS is in this operating state, you can call all DEC GKS functions except GKS$OPEN_GKS, GKS$CLOSE_GKS, or GKS$CLOSE_SEG. If you attempt to call these functions, DEC GKS produces an appropriate error message.

❹ When you open a segment, the DEC GKS operating state changes from from GKS$K_WSAC to GKS$K_SGOP (segment open). To accomplish this task, call GKS$CREATE_SEG and pass a segment name. The segment *name* is chosen by you for use in all references in the program to a specific segment. That segment is stored on all active workstations. To add output primitives to the segment, you need only call the desired DEC GKS output functions. Unless workstation independent segment storage (WISS) is open and active during segment creation, segments stored on workstations cannot

be copied from one workstation to another. You can only copy segments from WISS to an open or active workstation; you cannot copy a segment from any other type of workstation.

When you create a segment, DEC GKS creates a *segment state list.* The segment state list contains entries that specify the segment name, the set of associated workstations, the detectability of the segment, and so forth.

In the GKS$K_SGOP operating state, you can call all GKS functions except those that open or close DEC GKS, those that associate or copy the open segment to another workstation, those that attempt to change the state of the workstation, those that clear the workstation (GKS$CLEAR_WS), or those that create segments (GKS$CREATE_SEG). If you attempt to call those functions, DEC GKS generates an error message.

❺ When you close the open segment, the DEC GKS kernel changes the operating state from GKS$K_SGOP to GKS$K_WSAC.

❻ If the operating state is GKS$K_WSAC, and if you deactivate the last active workstation, the kernel changes the DEC GKS operating state from GKS$K_WSAC to GKS$K_WSOP.

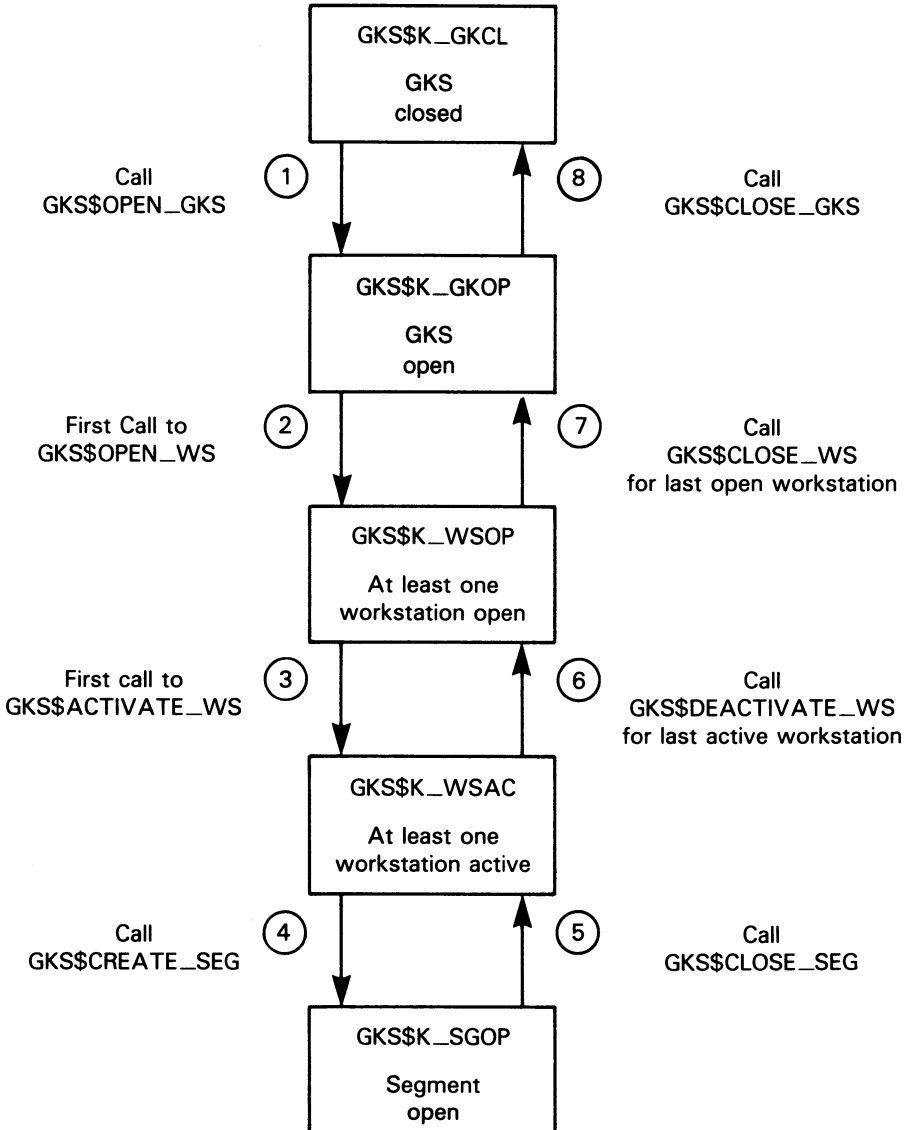
❼ Similarly, if you close the last open workstation, the kernel changes the DEC GKS operating state to GKS$K_GKOP.

❽ The final call in a single DEC GKS session should be to GKS$CLOSE_GKS; after the call, access to the DEC GKS environment is closed and your GKS session ends in an orderly fashion.

As you end your DEC GKS session, you must close an open segment (if one exists), close and deactivate workstations, and close DEC GKS, in the proper order. If you do not, your DEC GKS session does not end in an orderly fashion.

For example, if you fail to deactivate and to close an active workstation before ending your program, the workstation may not return control to the user, depending on the device.

Figure 4–1: GKS Operating States and Environment Control



4.2 Controlling the Workstation Display Surface

Depending on the type of device with which you are working, and depending on the values of certain entries in the workstation description tables and state lists, there may be times during program execution when the picture does not contain all of the changes previously requested by the application program. DEC GKS allows a workstation to delay the actions requested by a program in order to utilize most efficiently the capabilities of a workstation.

Output *deferral* is one workstation attribute that affects the rate of picture generation. By setting the deferral mode, you can *buffer* the generation of output images before transmission to the surface in order to improve overall rate of transmission, if a given workstation supports such buffering. Other times, you can release buffered output so that the display surface reflects the picture defined by the application.

4.2.1 Output Deferral

DEC GKS supports four deferral modes for its supported workstations. The deferral modes, in *increasing order of deferral*, are as follows:

- GKS$K_ASAP—Generates output As Soon As Possible.
- GKS$K_BNIG—Generates output Before the Next Interaction Globally.
- GKS$K_BNIL—Generates output Before the Next Interaction Locally.
- GKS$K_ASTI—Generates output at Some TIme (as defined by the workstation).

A local interaction happens on the workstation specified at the time of the surface update, and a global interaction happens on any open workstation. An *interaction* is a request for input using the DEC GKS input functions.

Depending on the capabilities of the workstation, it can defer output at any level up to the level specified in the call to GKS$SET_DEFER_STATE. If the workstation can defer output at the requested level, it does. If the workstation cannot defer output at the requested level, it defers output at the next supported lower level.

For example, if you specify GKS$K_ASAP in a call to GKS$SET_DEFER_STATE, the workstation must generate output as soon as possible. If you specify GKS$K_BNIG, the workstation can defer output at either GKS$K_ASAP or GKS$K_BNIG, depending on its capabilities. If you specify GKS$K_BNIL, the workstation can defer output on any level up to and including GKS$K_BNIL, depending on its capabilities. If you specify GKS$K_ASTI,

the workstation can defer output at any of the four levels, depending on its capabilities.

You can specify a suggested level of deferral by calling the function GKS$SET_DEFER_STATE. To determine the default deferral state of a given workstation type, you can call GKS$INQ_DEF_DEFER_STATE. To determine the current state of the deferral mode, you can call GKS$INQ_WS_DEFER_AND_UPDATE.

Writing applications with other graphics programs, you need to "flush the output buffer" in order to include all output in your picture. The DEC GKS equivalent of this action is to "release deferred output" (if there is any). To see if generated output has been deferred by the workstation, you call the function GKS$INQ_WS_DEFER_AND_UPDATE. To release deferred output without updating the screen in any other way, call the function GKS$UPDATE_WS and pass the argument GKS$K_POSTPONE_FLAG. For example, the VT125 and the VT240 defer output by default. If you are using those devices, you need to release deferred output if you want to place the current image on the workstation surface.

4.2.2 Implicit Surface Regenerations

Suppressed *implicit regeneration* of the currently generated output primitives is the second workstation attribute that can place the workstation surface out of date.

If you request a change to an output attribute bundle index, a change to a segment attribute, or a change to the current workstation window or viewport, the workstation can either make the change to the surface dynamically (GKS$K_IMM) or can implicitly regenerate the entire picture in order to comply with the requested change (GKS$K_IRG).

Whether a workstation makes the change dynamically or requires an implicit regeneration is a static capability of the particular workstation. You can call either the function GKS$INQ_DYN_MOD_SEG or GKS$INQ_DYN_MOD_WS to determine if a workstation can make a certain change immediately or if the picture must be implicitly regenerated.

If a workstation makes changes dynamically, then only the output primitives in the picture that are affected by the change are regenerated and the surface does not become out of date. For instance, for many of the supported workstations, a call to the function GKS$SET_COLOR_REP (refer to Chapter 6, Output Attribute Functions) changes color table entries dynamically.

When an implicit regeneration occurs, the workstation clears the surface, implements the change, and then redraws only the segments on the workstation surface. You lose all output primitives not contained in segments. For instance, for many of the supported workstations, a call to the function GKS$SET_PLINE_REP (refer to Chapter 6, Output Attribute Functions) causes an implicit regeneration on many workstations.

If a workstation makes changes by implicit regeneration, the workstation may or may not regenerate the workstation surface at that point in the program to implement the change. The *implicit regeneration mode* entry in the workstation state list specifies whether the workstation currently allows implicit regenerations, or if it suppresses them, leaving the workstation surface out of date. You can call the function GKS$INQ_WS_DEFER_AND_UPDATE to determine if the workstation is allowing regenerations (GKS$K_IRG_ALLOWED) or suppressing them (GKS$K_IRG_SUPPRESSED).

Many of the DEC GKS supported devices suppress implicit regenerations because of the possible loss of output primitives caused by an allowed regeneration. If you wish to change the implicit regeneration mode entry in the workstation state list, you can call the function GKS$SET_DEFER_STATE. Suppressing implicit regenerations allows you to make many changes to the picture without incurring the overhead of a regeneration for every change.

When you are ready to update the workstation surface, you can call GKS$UPDATE_WS, passing GKS$K_PERFORM_FLAG, to perform the single implicit regeneration. Rmember that if you call GKS$UPDATE_WS to force a surface regeneration, you lose all primitives not contained in segments.

4.2.3 Workstation Surface State List Entries

When controlling the workstation surface, you should be aware of the *display surface empty* and the *new frame action necessary at update* entries in the workstation state list.

Several of the control functions clear the workstation surface if the *display surface empty* entry is GKS$K_EMPTY. Under certain conditions, when you are working with different clipping rectangles and generalized drawing primitives (GDPs), the entry may contain GKS$K_NOTEMPTY when the surface is actually empty. In such situations, when the entry contains GKS$NOTEMPTY, the application program must decide whether or not there exists any "invisible" output to the workstation surface.

Also, you may wish to check the *new frame action necessary at update* entry to determine if an implicit regeneration will occur if you update the surface by calling GKS$UPDATE_WS (passing GKS$K_PERFORM_FLAG as an argument). If the new frame entry is GKS$K_NEWFRAME_NOTNECESSARY, then you can update the surface without the fear of losing primitives not contained in segments. If the new frame entry is GKS$K_NEWFRAME_NECESSARY, then a call to GKS$UPDATE_WS with the GKS$K_PERFORM_FLAG argument will cause an implicit regeneration, causing all primitives not contained in segments to be lost.

For more information, refer to Chapter 12, Inquiry Functions.

4.3 Control Inquiries

The following list presents the inquiry functions that you should use to obtain control function information when writing device-independent code:

GKS$INQ_ACTIVE_WS
GKS$INQ_DYN_MOD_WS_ATTB
GKS$INQ_LEVEL
GKS$INQ_OPEN_WS
GKS$INQ_OPERATING_STATE
GKS$INQ_WS_CATEGORY
GKS$INQ_WS_DEFER_AND_UPDATE
GKS$INQ_WS_MAX_NUM
GKS$INQ_WS_STATE
GKS$INQ_WS_TYPE
GKS$INQ_WSTYPE_LIST

For more information concerning device-independent programming, refer to the *DEC GKS User Manual.*

4.4 Function Descriptions

This section describes the control functions in detail.

ACTIVATE WORKSTATION

Operating States: WSOP, WSAC

Description

The function GKS$ACTIVATE_WS activates the specified workstation, allowing all subsequently generated output to be sent to the workstation.

You must open DEC GKS and you must open the workstation you wish to activate before calling GKS$ACTIVATE_WS. If the newly activated workstation is the only active workstation, DEC GKS changes the operating state from GKS$K_WSOP (at least one workstation open) to GKS$K_WSAC (at least one workstation active).

Syntax

GKS$ACTIVATE_WS *(workstation_id)*

GACWK *(workstation_id)*

gactivate *(workstation_id)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in this chapter).

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
6	GKS$_ERROR_6	GKS not in proper state: GKS shall be either in the state WSOP or in the state WSAC in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
29	GKS$_ERROR_29	Specified workstation is active in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
43	GKS$_ERROR_43	Maximum number of simultaneously active workstations would be exceeded in routine ****

Program Example

Example 4–1 illustrates the use of many of the DEC GKS control functions, including GKS$ACTIVATE_WS.

Example 4–1: GKS$CLEAR_WS and the GKS Control Functions

```
C     This program writes a text string to the screen, and then
C     clears the screen on the condition that it is not already clear.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID
      REAL START_X, START_Y, LARGER
      DATA START_X / 0.1 /, START_Y / 0.5 /, LARGER / 0.03 /,
     * WS_ID / 1 /

❶     CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
❷     CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
❸     CALL GKS$ACTIVATE_WS( WS_ID )

C     Write a line of text to the screen at a legible text height.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
❹     CALL GKS$TEXT( START_X, START_Y, 'GKS$CLEAR_WS should erase this')

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Clear the screen conditionally
❺     CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_CONDITIONALLY )

C     Release the GKS and workstation environments.
❻     CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ You must call the function GKS$OPEN_GKS to establish the DEC GKS environment. The logical name, SYS$ERROR, usually defaults to the standard output device (the terminal surface). DEC GKS translates the logical name to determine where to output error messages. You may find it convenient to pass a file specification so that you can review the generated error messages at any given time.

❷ When you initialize the specified workstation environment, you assign the workstation a numeric identifier (in this example, the number 1), a device name (in this example, DEC GKS translates the logical name GKS$CONID to determine the device name), and a workstation type (in this example, GKS$K_VT240 represents the VT241 device type). If you choose not to use DEC GKS constants and you wish to use default values, you can replace the constants with the value 0.

For more information concerning the constants used here, refer to GKS$OPEN_WS in this section or to Chapter 1, Introduction to DEC GKS.

❸ When activating a workstation using GKS$ACTIVATE_WS, use the workstation identifier that you specified as the first argument in the call to function GKS$OPEN_WS (in this example, the value 1).

❹ Using default windows and viewports, the function GKS$TEXT outputs a character string starting at the world coordinates (0.1, 0.5).

For more information concerning the coordinate systems, refer to Chapter 7, Transformation Functions. For more information concerning text output, refer to Chapter 5, Output Functions.

❺ The function GKS$CLEAR_WS, when passed GKS$K_CLEAR_CONDITIONALLY, clears the workstation under the condition that the surface contains output primitives. Since the previous function call wrote a character string to the workstation surface, this call clears the screen.

❻ When deactivating and closing the open workstation, pass the numeric workstation identifier previously specified in the call to GKS$OPEN_WS (in this example, the value 1).

CLEAR WORKSTATION

Operating States: WSOP, WSAC

Description

The function GKS$CLEAR_WS performs the tasks in the following order:

1. Generates all deferred output (see GKS$SET_DEFER_STATE in this section).
2. If the *display surface empty* workstation state list entry is GKS$K_NOTEMPTY, this function always clears the surface. If the surface is empty (GKS$K_EMPTY), then this function only clears the screen if you specify GKS$K_CLEAR_ALWAYS as an argument. If no other workstations are associated with the segment, the segment is deleted. For more information, refer to Chapter 9, Segment Functions.

After executing this function, DEC GKS sets the *display surface empty* workstation state list entry to GKS$K_EMPTY, the *workstation transformation update state* entry to GKS$K_NOTPENDING, and the *new frame necessary at update* state list entry to GKS$K_NEWFRAME_NOTNECESSARY.

Syntax

GKS$CLEAR_WS *(workstation_id, flag)*

GCLRWK *(workstation_id, control_flag)*

gclearws *(workstation_id, clearflag)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in this chapter).

flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument determines under which condition DEC GKS clears the screen. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_CLEAR_CONDITIONALLY	Clear if the surface is not empty.
1	GKS$K_CLEAR_ALWAYS	Clear the workstation.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
-37	DECGKS$_ERROR_NEG_37	Error in device handler during event flag allocation in routine ****
6	GKS$_ERROR_6	GKS not in proper state: GKS shall be either in the state WSOP or in the state WSAC in routine ****

Error Number	Completion Status Code	Message
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****

Program Example

Refer to Example 4–1 in this section for a program example using a call to GKS$CLEAR_WS.

CLOSE GKS

Operating States: GKOP

Description

The function GKS$CLOSE_GKS releases the DEC GKS buffers, closes the error log file, and deletes it if empty. GKS$CLOSE_GKS releases the DEC GKS description table, the DEC GKS state list, and the workstation description tables. A call to GKS$CLOSE_GKS must end each DEC GKS session.

You must call both GKS$DEACTIVATE_WS for each active workstation and GKS$CLOSE_WS for each open workstation before you call GKS$CLOSE_GKS. If you do not, DEC GKS logs an error message.

A call to GKS$CLOSE_GKS changes the DEC GKS operating state from GKS$K_GKOP (GKS open) to GKS$K_GKCL (GKS closed).

Syntax

GKS$CLOSE_GKS *()*

GGKOP *()*

gclosegks *()*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
2	GKS$_ERROR_2	GKS not in proper state; GKS shall be in the state GKOP in routine ****

Program Example

Refer to Example 4–1 in this section for a program example using a call to GKS$CLOSE_GKS.

CLOSE WORKSTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$CLOSE_WS updates the workstation (equivalent to a call to GKS$UPDATE_WS with the GKS$K_PERFORM_FLAG argument), closes a workstation opened by a previous call to GKS$OPEN_WS, and releases a specified workstation's state list. GKS$CLOSE_WS deassigns the channel used for input/output to the device and removes the workstation from the set of open workstations in the DEC GKS state list.

If you call this function to close the last open workstation, this function changes the DEC GKS operating state from GKS$K_WSOP (at least one workstation open) to GKS$K_GKOP (GKS open).

Be sure to deactivate a workstation with a call to GKS$DEACTIVATE_WS before you attempt to close a workstation with GKS$CLOSE_WS. If you do not, DEC GKS logs an appropriate error message.

Syntax

GKS$CLOSE_WS *(workstation_id)*

GCLWK *(workstation_id)*

gclosews *(workstation_id)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in this chapter).

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
29	GKS$_ERROR_29	Specified workstation is active in routine ****
147	GKS$_ERROR_147	Input queue has overflowed in routine ****

Program Example

Refer to Example 4–1 in this section for a program example using a call to GKS$CLOSE_WS.

DEACTIVATE WORKSTATION

Operating States: WSAC

Description

The function GKS$DEACTIVATE_WS deactivates a specific workstation so that subsequent output will not be sent to that workstation. A call to this function removes the workstation from the set of active workstations in the DEC GKS state list. Segments stored on the workstation are retained.

If a call to this function deactivates the last active workstation, this function changes the DEC GKS operating state from GKS$K_WSAC (at least one workstation active) to GKS$K_WSOP (at least one workstation open).

You must deactivate a workstation before you can close that workstation. Also, you must deactivate and close all workstations (if applicable) before you can close DEC GKS. Otherwise, DEC GKS logs an appropriate error message.

Syntax

GKS$DEACTIVATE_WS *(workstation_id)*

GDAWK *(workstation_id)*

gdeactivate *(workstation_id)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in this chapter).

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
3	GKS$_ERROR_3	GKS not in proper state: GKS shall be in the state WSAC in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
30	GKS$_ERROR_30	Specified workstation is not active in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****

Program Example

Refer to Example 4–1 in this section for a program example using a call to GKS$DEACTIVATE_WS.

ESCAPE

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$ESCAPE provides a method for individual DEC GKS implementations to access capabilities of a specific workstation that are not fully utilized by other functions. For example, the DEC GKS implementation uses this function call to produce a hardcopy dump of a VT125/VT240 terminal screen or to set the plotter pen speed on an LVP16 workstation.

DEC GKS Device Specifics Reference Manual describes the level of support for the DEC GKS escape functions. For more information concerning the available escapes, refer to the *DEC GKS Device Specifics Reference Manual.*

SYNTAX

GKS$ESCAPE *(function_id, in_data_record, in_record_size, out_data_record, out_record_size, total_record_size)*

GESC *(fun_id, dim_idr, idr, max_odr, len_odr, odr)*

gescape *(function, indata, bufsize, outdata, escout_size)*

Arguments

function_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the escape function identifier.

in_data_record

data type: **address (record)**
access: **read-only**
mechanism: **by reference**

This argument is a pointer to the input data record buffer. For more information concerning the structure of the input data record, refer to Chapter 1, Introduction to DEC GKS.

in_record_size

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the size of the input data record in bytes. This argument should be the exact size of the required input data record.

out_data_record

data type: **address (record)**
access: **modifiable**
mechanism: **by reference**

This argument is a pointer to the output data record buffer. For more information concerning the structure of the output data record, refer to Chapter 1, Introduction to DEC GKS.

our_record_size

data type: **integer**
access: **modifiable**
mechanism: **by reference**

On input, this argument contains the size of the output data record buffer in bytes. On output, DEC GKS writes the amount of the buffer actually containing the output data record. If the argument total_record_size is larger than out_record_size, then you know that DEC GKS truncated the output data record when writing to the buffer.

If this argument is the value 0, DEC GKS only checks for errors and then writes the size of the output data record to total_record_size; the escape is not performed. In this way, you can obtain the actual size of the data record to compare it to your buffer space.

total_record_size

data type: **integer**
access: **write-only**
mechanism: **by reference**

This argument is the total size of the output data record in bytes. If the total size of the output data record does not match the size of the output buffer, you know that the record was either truncated to fit in the allocated space or was smaller than the allocated space.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
180	GKS$_ERROR_180	Specified escape function is not supported in routine ****
181	GKS$_ERROR_181	Specified escape function identification is invalid in routine ****
182	GKS$_ERROR_182	Contents of escape data record are invalid in routine ****

Program Example

Example 4–2 illustrates the use of the function GKS$ESCAPE. To achieve the same effects, you should connect a printer to the printer port of the VT241 terminal. Following the program example, Figure 4–2 illustrates the program's effect on a VT241 workstation.

Example 4–2: Using the Escape Function

```
C     This program outputs a tall, thin house from a VT240 screen to
C     an attached printer.
      IMPLICIT NONE
      INCLUDE 'GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, IN_DATA( 7 ),
     * IN_SIZE, OUT_DATA( 7 ), OUT_RECORD_SIZE,
     *TOTAL_RECORD_SIZE, LIST_INTS( 1 ), LIST_INTS_PTR
      REAL PX ( 9 ), PY ( 9 )
❶     DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
❷     DATA NUM_POINTS / 9 /, IN_SIZE / 16 /, WS_ID / 1 /,
     * OUT_RECORD_SIZE / 28 /

      EQUIVALENCE( IN_DATA( 4 ), LIST_INTS_PTR )
      LIST_INTS_PTR = %LOC( LIST_INTS )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❸     CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
C     Initialize escape data record...
❹     IN_DATA( 1 ) = 1
      IN_DATA( 2 ) = 0
      IN_DATA( 3 ) = 0
      LIST_INTS( 1 ) = WS_ID

❺     CALL GKS$ESCAPE( GKS$K_ESC_PRINT, IN_DATA, IN_SIZE,
     * OUT_DATA, OUT_RECORD_SIZE,
     * TOTAL_RECORD_SIZE )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ These arrays contain the house's X and Y world coordinates. For example, the first element of the array PX is the X value of the first point, and the first element of the array PY is the Y value of the first point (.4, .1). This program uses the default world coordinate plane with the starting point (0, 0), the maximum X value 1.0, and the maximum Y value 1.0.

❷ You must correctly initialize the data record size arguments. According to the description of the GKS$K_ESC_PRINT escape, the input data record is 16 bytes long (4 longword components). The initial size of the output data record is 28 bytes long (GKS$ESCAPE ignores the output data record for GKS$K_ESC_PRINT).

If you initialize the argument OUT_RECORD_SIZE to be the value 0, DEC GKS does not perform the escape, but instead returns the length of the output data record to the argument TOTAL_RECORD_SIZE. This functionality is useful when you are not sure if your output data record buffer is large enough.

❸ This code outputs a tall, thin house to the VT241 terminal screen.

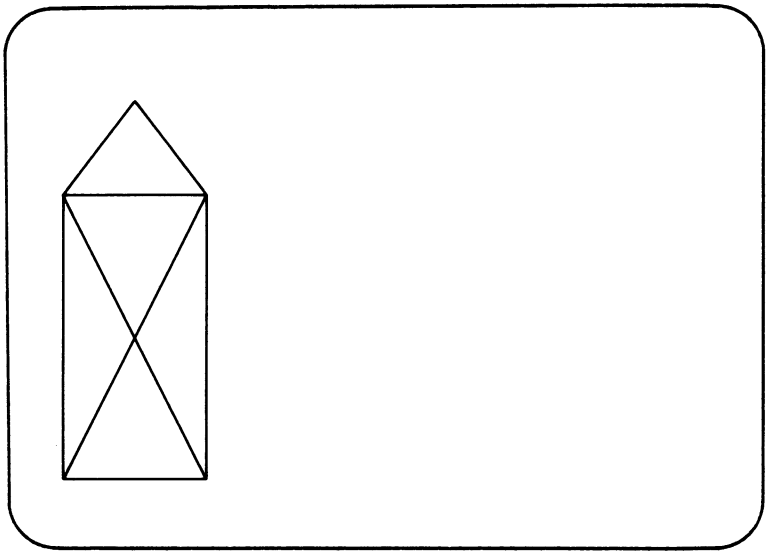
❹ The input data record passed to GKS$ESCAPE must contain the workstation identifier of the device containing the picture to be printed. In this example, the variable WS_ID represents the VT241 screen.

For a complete description of the DEC GKS standard escape data record format, refer to Chapter 1, Introduction to DEC GKS. For a complete description of the DEC GKS supported escapes and their data record requirements, refer to Appendix I, DEC GKS GDPs and Escapes.

❺ The call to GKS$ESCAPE outputs the picture to the attached printer.

Figure 4–2 shows the screen of a VT241 terminal after the program has run to completion.

Figure 4–2: Using the Escape Function—VT241



ZK-5043-86

MESSAGE

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$MESSAGE allows an application program to deliver a message to the user at an implementation-dependent location on the workstation surface, or on a separate device associated with the workstation. (For example, you may wish to send a message to the user stating the need to change the paper in the plotter before you regenerate the picture.)

For information on the workstation-specific capabilities of GKS$MESSAGE, refer to the *DEC GKS Device Specifics Reference Manual.*

SYNTAX

GKS$MESSAGE *(workstation_id, message)*

GMSG *(workstation_id, message)*

GMSGS - subset *(workstation_id, l_message, message)*

gmessage *(workstation_id, message)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in this chapter).

MESSAGE

message

data type: **string**
access: **read-only**
mechanism: **by descriptor**

This argument is the text of the message to be delivered to the specified workstation.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
-39	DECGKS$_ERROR_NEG_39	Descriptor is not acceptable in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
101	GKS$_ERROR_101	Number of points is invalid in routine ****

Program Example

Example 4–3 illustrates the use of the function GKS$MESSAGE. To achieve the same effects, you need to do the following:

1. Determine the device connection identifier of the printer you want to use as a workstation. The printer must be attached to your host system, not to

your terminal. The DIGITAL Command Language (DCL) command SHOW DEVICE may assist you.

2. Allocate that device for your use using the DCL command ALLOCATE. (You may need special privileges to allocate a device.)
3. Use the DCL command SHOW TERMINAL for the allocated device, to determine whether the baud rate, parity, and other terminal characteristics match the communications settings on the printer. For more information on these settings, refer to the programming guide for the printer.
4. Use the DCL command DEFINE to define GKS$CONID as the connection identifier for the allocated device, and GKS$WSTYPE as the type of printer you are using. For more information concerning the possible workstation types, refer to the appropriate appendix in this manual.

Following the program example, Figure 4–3 illustrates the program's effect on a VT241 workstation.

Example 4–3: Sending a Message to the User

```
C     This program outputs a "tall, thin house" to
C     an LA100 and then prints a message on the VT241 terminal
C     screen.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_SCREEN, WS_PRINTER, NUM_POINTS
      REAL PX ( 9 ), PY ( 9 )
❶     DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, WS_PRINTER / 1 /, WS_SCREEN / 2 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_PRINTER, GKS$K_CONID_DEFAULT,
     * GKS$K_WSTYPE_DEFAULT )
      CALL GKS$ACTIVATE_WS( WS_PRINTER )
❷     CALL GKS$OPEN_WS( WS_SCREEN, 'TT:', GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_SCREEN )

❸     CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
```

(continued on next page)

Example 4–3 (Cont.): Sending a Message to the User

```
❹       CALL GKS$MESSAGE( WS_SCREEN, 'I just finished printing the house')
C       Release deferred output. Pause.  Type RETURN when you are finished
C       viewing the picture.
        CALL GKS$UPDATE_WS( WS_SCREEN, GKS$K_POSTPONE_FLAG )
        READ(5,*)

        CALL GKS$DEACTIVATE_WS( WS_PRINTER )
        CALL GKS$CLOSE_WS( WS_PRINTER )
        CALL GKS$DEACTIVATE_WS( WS_SCREEN )
        CALL GKS$CLOSE_WS( WS_SCREEN )
        CALL GKS$CLOSE_GKS()
        END
```

The following numbers correspond to the numbers in the previous example:

❶ These arrays contain the house's X and Y world coordinates. For example, the first element of the array PX is the X value of the first point, and the first element of the array PY is the Y value of the first point (.4, .1). This program uses the default world coordinate plane with the starting point (0, 0), the maximum X value 1.0, and the maximum Y value 1.0.

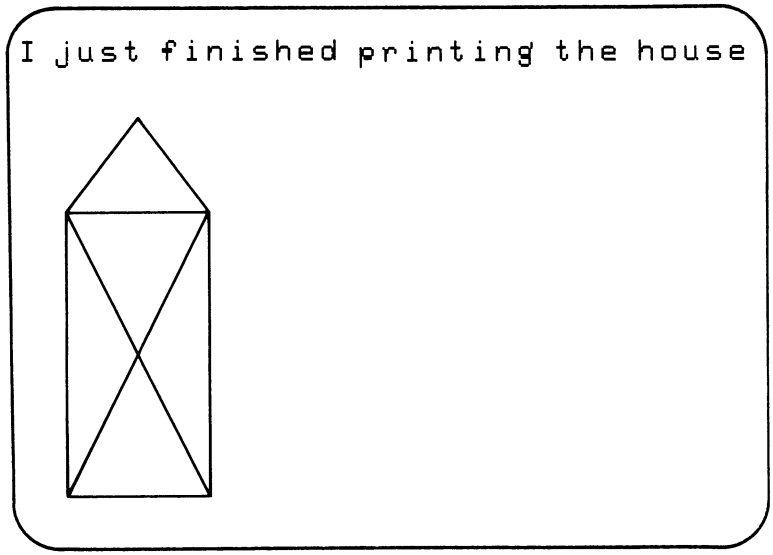
❷ The logical name TT defaults to the device connection of your terminal, which in this example is a VT241.

❸ This code outputs a tall, thin house to the printer.

❹ The call to GKS$MESSAGE outputs a message to the VT241 screen telling the user that the program printed the picture of the house.

Figure 4–3 shows the screen of a VT241 terminal after the program has run to completion.

Figure 4–3: Sending the User a Message—VT241



OPEN GKS

Operating States: GKCL

Description

The function GKS$OPEN_GKS permits subsequent access to the DEC GKS state list, DEC GKS description table, and the workstation description tables.

GKS$OPEN_GKS changes the DEC GKS operating state from GKS$K_GKCL (GKS closed) to GKS$K_GKOP (GKS open). The error state list entry *error file* is set to the value passed as an argument to this function.

When using DEC GKS, you usually call GKS$OPEN_GKS first. All functions except emergency close, error handling, error logging, GKS$OPEN_GKS, and GKS$INQ_OPERATING_STATE require at least the GKS$K_GKOP operating state.

Syntax

GKS$OPEN_GKS *(error_file, [, memory])*

GOPKS *(err_file, [, buffer])*

gopengks *(errfile, memory)*

Arguments

error_file

data type:	**string**
access:	**read-only**
mechanism:	**by descriptor**

This argument is either a logical name or a physical name of a device or file that points to the error log file. For information on how GKS handles errors, refer to Chapter 11, Error-Handling Functions.

NOTE

If you pass the value 0, DEC GKS uses the translation of the logical name SYS$ERROR as the error file.

memory

data type: **integer**
access: **read-only**
mechanism: **by reference**

To maintain compatibility with the GKS standard, GKS$OPEN_GKS accepts an optional second argument to indicate the amount of memory units available for use by DEC GKS. If provided, DEC GKS ignores this argument.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
-39	DECGKS$_ERROR_NEG_39	Descriptor is not acceptable in routine ****
-97	DECGKS$_ERROR_NEG_97	Internal GKS error: Insufficient buffer size for translated logical name in routine ****
-98	DECGKS$_ERROR_NEG_98	Internal GKS error: Too many translations of logical name in routine ****
1	GKS$_ERROR_1	GKS not in proper state: GKS shall be in the state GKCL in routine ****
200	GKS$_ERROR_200	Specified error file is invalid in routine ****

Program Example

Refer to Example 4–1 in this section for a program example using a call to GKS$OPEN_GKS.

OPEN WORKSTATION

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$OPEN_WS initializes a workstation for use by DEC GKS, permitting subsequent access to the specified workstation's state list. The function associates the workstation identifier with a particular device of a specified type, and initializes the workstation.

If establishing the first open workstation, GKS$OPEN_WS changes the DEC GKS operating state from GKS$K_GKOP (GKS open) to GKS$K_WSOP (at least one workstation open).

GKS$OPEN_WS clears the display surface of previously generated images. You must call this function, followed by a call to GKS$ACTIVATE_WS, before you attempt to generate output to this workstation.

Syntax

GKS$OPEN_WS *(workstation_id, device_connection_id, workstation_type)*

GOPWK *(workstation_id, con_id, workstation_type)*

gopenws *(workstation_id, conid, workstation_type)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation. You choose whichever **nonnegative** integer value your application requires.

device_connection_id

data type:	**string**
access:	**read-only**
mechanism:	**by descriptor**

This argument is the device connection identifier that is associated with a particular device attached to the host system. This argument can be a logical name, a DEC GKS constant value, or a VMS file specification.

As an option, you can pass either the DEC GKS constant GKS$K_CONID_DEFAULT, or the value 0, by reference. (If you use VAX BASIC, you can pass a null string, by reference.) If you use this option, DEC GKS translates the logical name GKS$CONID and uses the translation as the name of the device. This feature aids program flexibility; each time you execute your program, you can use the DIGITAL Command Language commands DEFINE or ASSIGN to define GKS$CONID to be the device connection of your choice. For more information, refer to Chapter 1, Introduction to DEC GKS.

If the translation of GKS$CONID is not valid, DEC GKS uses the logical name TT as the device. The logical name TT is equivalent to your default terminal connection.

Using certain output-only workstation types, you can specify a file name for this argument. When you specify a file name, DEC GKS places the graphical information into the specified file in the current (or specified) directory. You can then print or type the file on the workstation. If you omit the file extension, DEC GKS uses the default extension .LIS. Otherwise, DEC GKS uses the file name as specified. To determine whether your workstation supports an output-only workstation type that would allow you to specify a file name for this argument, refer to the *DEC GKS Device Specifics Reference Manual.*

NOTE

If you use GKS$OPEN_WS to create or to read a metafile, DEC GKS uses file names for this argument exactly as specified, without applying a default extension. For more information concerning metafiles, refer to Chapter 10, Metafile Functions.

workstation_type

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is an integer value that specifies the workstation type. To review the list of possible workstation type values, refer to Appendix A, DEC GKS Supported Workstations. For more information concerning valid workstation type bitmasks for a given device, refer to the appropriate *DEC GKS Device Specifics Reference Manual.*

As an option, you can pass either the GKS constant GKS$K_WSTYPE_DEFAULT, or the value 0, by reference. If you use this option, GKS translates the logical name GKS$WSTYPE and uses the translation as the name of the workstation type. This feature aids program flexibility; each time you execute your program, you can use the DIGITAL Command Language commands DEFINE or ASSIGN to define GKS$WSTYPE to be the workstation type of your choice. For more information, refer to Chapter 1, Introduction to DEC GKS.

If GKS$WSTYPE translates to the value 0, DEC GKS sets the default workstation type to GKS$K_VT240BW (on a large VAX) or to GKS$K_VSII (on a VAXstation).

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
-30	DECGKS$_ERROR_NEG_30	Cannot load workstation handler: error during image activation in routine ****
-31	DECGKS$_ERROR_NEG_31	Cannot load graphics handler: invalid DFT in routine ****

Error Number	Completion Status Code	Message
-35	DECGKS$_ERROR_NEG_35	Kernel has detected an unexpected error from a graphics handler in routine ****
-39	DECGKS$_ERROR_NEG_39	Descriptor is not acceptable in routine ****
-97	DECGKS$_ERROR_NEG_97	Internal GKS error: Insufficient buffer size for translated logical name in routine ****
-98	DECGKS$_ERROR_NEG_98	Internal GKS error: Too many translations of logical name in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
21	GKS$_ERROR_21	Specified connection identifier is invalid in routine ****
22	GKS$_ERROR_22	Specified workstation type is invalid in routine ****
23	GKS$_ERROR_23	Specified workstation type does not exist in routine ****
24	GKS$_ERROR_24	Specified workstation is open in routine ****
26	GKS$_ERROR_26	Specified workstation cannot be opened in routine ****
28	GKS$_ERROR_28	Workstation Independent Segment Storage is already open in routine ****
42	GKS$_ERROR_42	Maximum number of simultaneously open workstations would be exceeded in routine ****

Program Example

Refer to Example 4–1 in this section for a program example using a call to GKS$OPEN_WS.

REDRAW ALL SEGMENTS ON WORKSTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$REDRAW_SEG_ON_WS clears the screen and redraws all defined, visible segments. The function performs the following tasks in order:

1. Generates all deferred output (see GKS$SET_DEFER_STATE in this section).
2. If the *display surface empty* workstation state list entry is GKS$K_NOTEMPTY, this function clears the surface.
3. Places into effect pending workstation transformations.
4. Redisplays all visible segments that existed on the workstation surface before the screen was cleared. All output not contained in segments is lost.

After executing this function, DEC GKS sets the *workstation transformation update state* entry to GKS$K_NOTPENDING, and the *new frame necessary at update* state list entry to GKS$K_NEWFRAME_NOTNECESSARY.

NOTE

You should use this function if you need to redraw the picture regardless of the status of the *new frame necessary at update* entry. Otherwise, use GKS$UPDATE_WS.

Syntax

GKS$REDRAW_SEG_ON_WS *(workstation_id)*

GRSGWK *(workstation_id)*

gredrawsegws *(workstation_id)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in this chapter).

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Storage in routine ****

Program Example

Example 4–4 illustrates the use of the function GKS$REDRAW_SEG_ON_WS. Following the program example, Figure 4–4 illustrates the program's effect on a VT241 workstation.

Example 4–4: Redrawing Segments

```
C     This program creates a segment and then calls
C     GKS$REDRAW_SEG_ON_WS.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, TRIANGLE
      REAL PX( 4 ), PY( 4 ), LARGER
      DATA WS_ID / 1 /, NUM_POINTS / 4 /, TRIANGLE / 1 /,
     * LARGER / 0.02 /
❶     DATA PX  /.1, .9, .1, .1 /
      DATA PY  /.1, .9, .9, .1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Set the current bundle index to 1.
      CALL GKS$SET_PLINE_INDEX( PLINE_INDEX )

❷     CALL GKS$CREATE_SEG( TRIANGLE )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG( )

C     Make the text easier to see and then generate the string.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.1, 0.3,
     * 'THIS IS NOT PART OF THE SEGMENT' )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

❸     CALL GKS$REDRAW_SEG_ON_WS( WS_ID )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

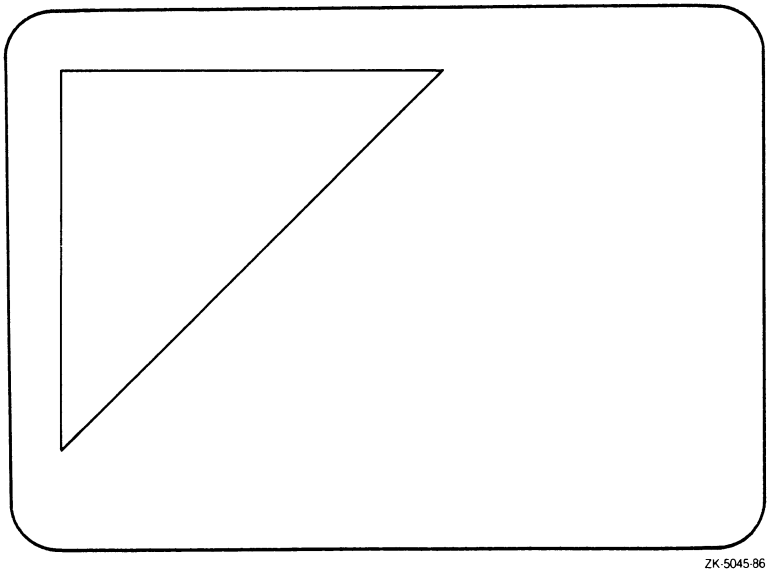
❶ These arrays contain the polygon's X and Y world coordinates. For example, the first element of array PX is the X value of the first point and the first element of array PY is the Y value. The first point of the polygon is (.1, .1).

❷ This code creates a segment that contains a triangle. The code also generates a text string to the screen that is independent of the segment.

❸ The call to GKS$REDRAW_SEG_ON_WS redraws the triangle, but does not redraw the text string since the string is not part of a segment.

Figure 4–4 shows the screen of the VT241 terminal after the program has run to completion.

Figure 4–4: Redrawing Segments—VT241



SET DEFERRAL STATE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_DEFER_STATE sets the workstation state list entries *deferral mode* and *implicit regeneration mode*. Using this function, you can allow a workstation to defer output, or you can either suppress or allow implicit regenerations (see Section 4.2.1 for detailed information).

The deferral mode specifies the rate of output generation. Depending on the capabilities of the workstation, it can defer output at any level up to the level specified in the call to GKS$SET_DEFER_STATE. If the workstation can defer output at the requested level, it does. If the workstation cannot defer output at the requested level, it defers output at the next supported lower level.

For example, if you specify GKS$K_ASAP in a call to GKS$SET_DEFER_STATE, the workstation must generate output as soon as possible. If you specify GKS$K_BNIG, the workstation can defer output at either GKS$K_ASAP or GKS$K_BNIG, depending on its capabilities. If you specify GKS$K_BNIL, the workstation can defer output on any level up to and including GKS$K_BNIL, depending on its capabilities. If you specify GKS$K_ASTI, the workstation can defer output at any of the four levels, depending on its capabilities. (For more information concerning the definitions of the constants discussed in this paragraph, refer to the deferral_mode argument description.)

The implicit regeneration mode determines whether implicit regenerations are allowed (GKS$K_IRG_ALLOWED) or suppressed (GKS$K_IRG_SUPPRESSED). If you allow implicit regenerations, any pending or subsequent surface change requiring regeneration (possibly output bundle index changes, segment attribute changes, or workstation transformation changes) occurs at the time of request. If you suppress regenerations, changes requiring regenerations place the screen out of date (DEC GKS sets the *new frame necessary at update* entry in the workstation state list to GKS$K_NEWFRAME_NECESSARY).

SET DEFERRAL STATE

By suppressing implicit regenerations, you can make all necessary changes without altering the workstation surface. Then, when you have requested all changes, call GKS$UPDATE_WS to perform all of suppressed actions in a single regeneration of the surface.

NOTE

When regenerating the surface of the workstation, DEC GKS clears the surface before redrawing only the visible segments. All output primitives not contained in segments are lost.

Syntax

GKS$SET_DEFER_STATE *(workstation_id, deferral_mode, regeneration_mode)*

GSDS *(workstation_id, def_mode, reg_mode)*

gsetdeferst *(workstation_id, defmode, irgmode)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in this chapter).

deferral_mode

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument specifies the maximum allowable deferral mode. The device implements the highest supported mode up to, and including, this specified mode. The argument can be any of the following values or constants.

Value	Constant	Description
0	GKS$K_ASAP	Generate images as soon as possible.
1	GKS$K_BNIG	Generate images before the next interaction globally (before you request input from any open workstation).
2	GKS$K_BNIL	Generate images before the next interaction locally (before the next call for input from the specified workstation).
3	GKS$K_ASTI	Generate images at some time. The exact time is determined by the workstation.

regeneration_mode

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument specifies the implicit regeneration mode. The argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_IRG_SUPPRESSED	Image regeneration is suppressed.
1	GKS$K_IRG_ALLOWED	Image regeneration is allowed.

Be aware that if you call GKS$SET_DEFER_STATE and pass GKS$K_IRG_ALLOWED, you force the device to implicitly regenerate the surface at the time of the function call. When DEC GKS implicitly regenerates a workstation surface, it clears the surface and redraws all visible segments stored on that workstation. You lose any output primitives not stored in segments.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
-26	DECGKS$_ERROR_NEG_26	Invalid value specified for deferral mode in routine ****
-27	DECGKS$_ERROR_NEG_27	Invalid value specified for regeneration mode in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Storage in routine ****

Program Example

Example 4–5 illustrates the use of the function, GKS$SET_DEFER_STATE. Following the program example, Figure 4–5 illustrates the program's effect on a VT241 workstation.

Example 4–5: Suppressing Implicit Regeneration

```
C     This program changes the color of a triangle from
C     the default color green, to yellow. Then, the line
C     changes from solid to dashed, and from yellow to blue.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, GREEN, NUM_POINTS, NUM_FLAGS, INCR,
     * PLINE_INDEX, TRIANGLE, BLUE
      REAL PX( 4 ), PY( 4 ), LARGER, RED_INTENS,
     * GREEN_INTENS, BLUE_INTENS
      DATA WS_ID / 1 /, NUM_FLAGS / 13 /, GREEN / 1 /, NUM_POINTS / 4 /,
     * RED_INTENS / 1.0000 /, GREEN_INTENS / 1.0000 /, BLUE / 3 /,
     * BLUE_INTENS / 0.4200 /, PLINE_INDEX / 1 /, LARGER / 1.0 /,
     * TRIANGLE / 1 /
      INTEGER FLAGS( 13 )
(1)   DATA PX  /.1, .9, .1, .1 /
      DATA PY  /.1, .9, .9, .1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

(2)   CALL GKS$SET_DEFER_STATE( WS_ID, GKS$K_ASAP,
     * GKS$K_IRG_SUPPRESSED )

(3)   DO 100 INCR = 1, NUM_FLAGS, 1
      FLAGS( INCR ) = GKS$K_ASF_BUNDLED
100   CONTINUE
      CALL GKS$SET_ASF( FLAGS )

(4)   CALL GKS$CREATE_SEG( TRIANGLE )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG( )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

(5)   CALL GKS$SET_COLOR_REP( WS_ID, GREEN,
     * RED_INTENS, GREEN_INTENS, BLUE_INTENS )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
```

(continued on next page)

Example 4–5 (Cont.): Suppressing Implicit Regeneration

```
❻      CALL GKS$SET_PLINE_REP( WS_ID, PLINE_INDEX,
      * GKS$K_LINETYPE_DASHED, LARGER, BLUE )

❼      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

       CALL GKS$DEACTIVATE_WS( WS_ID )
       CALL GKS$CLOSE_WS( WS_ID )
       CALL GKS$CLOSE_GKS()
       END
```

The following numbers correspond to the numbers in the previous example:

❶ These arrays contain the polygon's X and Y world coordinates. For example, the first element of array PX is the X value of the first point and the first element of array PY is the Y value. The first point of the polygon is (.1, .1).

❷ The call to GKS$SET_DEFER_STATE does not defer output, but it does suppress all implicit regenerations.

❸ This code initializes the array that designates attributes as either bundled or individually specified. The DO loop initializes all thirteen members of the array with the constant GKS$K_ASF_BUNDLED. The variable INCR increments the array; the variable NUM_FLAGS contains the number of elements in the array (13).

For more information, refer to GKS$SET_ASF in Chapter 6, Output Attribute Functions.

❹ This code places the triangle in a segment.

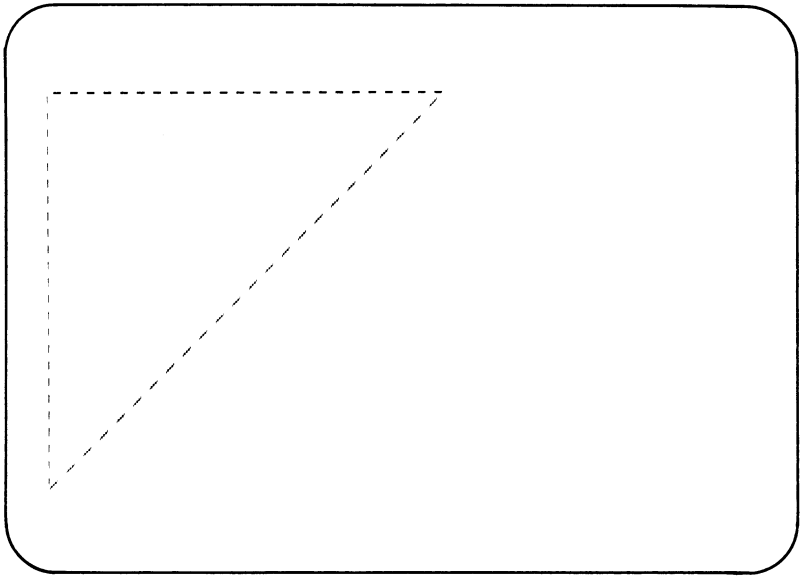
❺ This code changes the color representation from green to yellow. Changing the color representation of a color index value does not necessitate an implicit regeneration on a VT241; the change happens on the surface immediately.

❻ Notice that DEC GKS suppresses the change to the polyline representation since, on a VT241, changing the polyline requires an implicit regeneration (which is suppressed through the call to GKS$SET_DEFER_STATE).

❼ Once you call GKS$UPDATE_WS and pass the argument GKS$K_PERFORM_FLAG, DEC GKS clears the surface and regenerates the segment with the new bundled polyline attributes.

Figure 4–5 shows the screen of the VT241 terminal after the program has run to completion.

Figure 4–5: Suppressing Implicit Regeneration—VT241



ZK-5218-86

UPDATE WORKSTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$UPDATE_WS generates all deferred output for the specified workstation without first clearing the screen. Then, if the *new frame necessary at update* entry in the workstation state list is GKS$K_NEWFRAME_NECESSARY, and if you specify GKS$K_PERFORM_FLAG to this function, GKS$UPDATE_WS performs the following tasks in order:

1. Clears the screen if the *display surface empty* entry in the workstation state list is GKS$K_NOTEMPTY.
2. Places into effect pending workstation transformations.
3. Redisplays all visible segments that were stored on the workstation. All output primitives not contained in segments are lost.

After executing the actions listed previously, DEC GKS sets the *display surface empty* workstation state list entry to GKS$K_EMPTY or to GKS$K_NOTEMPTY according to the current state of the workstation surface, the *workstation transformation update state* entry to GKS$K_NOTPENDING, and the *new frame necessary at update* state list entry to GKS$K_NEWFRAME_NOTNECESSARY. The deferral and regeneration mode entries in the workstation state list have the same values as they did before the call to GKS$UPDATE_WS.

However, if at the call to GKS$UPDATE_WS the *new frame necessary at update* entry in the workstation state list is GKS$K_NEWFRAME_NOTNECESSARY, or if you specify GKS$K_POSTPONE_FLAG as an argument to this function, GKS$UPDATE_WS initiates only the transmission of any deferred output and will continue to suppress implicit regenerations. Again, the *deferral mode* and *regeneration mode* entries in the workstation state list have the same values as they did before the call to GKS$UPDATE_WS.

Syntax

GKS$UPDATE_WS *(workstation_id, flag)*

GUWK *(workstation_id, reg_flag)*

gupdate *(workstation_id, regenflag)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in this chapter).

flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument establishes the implicit regeneration mode The argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_POSTPONE_FLAG	Suppress regeneration of images.
1	GKS$K_PERFORM_FLAG	Perform regeneration.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the ERROR state in routine ****
-25	DECGKS$_ERROR_NEG_25	Invalid value specified for update workstation flag in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Storage in routine ****

Program Example

For a program example using a call to GKS$UPDATE_WS, refer to Example 4–5 in this section.

For an example of this function updating a workstation surface after a change to a workstation viewport and window, refer to Chapter 7, Transformation Functions.

Chapter 5

Output Functions

The DEC GKS output functions generate the basic components, or **primitives**, of all graphical pictures. The output functions are divided into the following categories:

Category	GKS Functions
Draw connected lines.	GKS$POLYLINE
Mark locations with symbols.	GKS$POLYMARKER
Draw text.	GKS$TEXT
Fill a polygon.	GKS$FILL_AREA
Color cells of a rectangle.	GKS$CELL_ARRAY
Draw generalized drawing primitive.	GKS$GDP

When you generate primitives on the workstation surface, you should be aware of the following:

- DEC GKS operating state
- DEC GKS coordinate systems
- Transformations
- Clipping
- Deferred transformations and output

The following sections describe these issues related to output, and point to the appropriate chapters in this manual that discuss the topics in full detail.

5.1 Output and the DEC GKS Operating State

As you call control functions, DEC GKS allows access to certain tables and lists. You can never call a DEC GKS function that requires access to a table or list that has not yet been made available. To determine which tables and lists are accessible, and which DEC GKS functions you can call at a given point in the application program, DEC GKS maintains an *operating state*.

The DEC GKS operating states are as follows:

- GKS$K_GKCL—DEC GKS is closed.
- GKS$K_GKOP—DEC GKS is open.
- GKS$K_WSOP—At least one workstation is open.
- GKS$K_WSAC—At least one workstation is active.
- GKS$K_SGOP—A segment is open.

To call any of the output functions described in this chapter, the DEC GKS operating state must be GKS$K_WSAC or GKS$K_SGOP. To place DEC GKS into the GKS$K_WSAC operating state, you need to do the following:

- Open DEC GKS (by calling GKS$OPEN_GKS).
- Open at least one workstation (by calling GKS$OPEN_WS).
- Activate at least one workstation (by calling GKS$ACTIVATE_WS).

If you call an output function, DEC GKS generates the primitive on all active workstations. If you call an output function during the GKS$K_SGOP operating state, then the output primitive becomes part of a segment. (For complete information concerning segments, refer to Chapter 9, Segment Functions.)

If you wish to output to an active workstation, the workstation must be of type GKSK_WSCAT_OUTPUT, GKSK_WSCAT_OUTIN, or GKS$K_WSCAT_MO. Only workstations of those categories support image generation. GKS$K_WSCAT_OUTPUT and GKS$K_WSCAT_OUTIN workstations generate output primitives on the workstation surface; GKS$K_WSCAT_MO workstations store information about the function call in a file. For more information concerning metafiles, refer to Chapter 10, Metafile Functions. For more information concerning workstation categories or the DEC GKS operating states, refer to Chapter 4, Control Functions.

5.2 Output Attributes

All of the output primitives have *attributes* that are stored in the DEC GKS state list. Attributes are properties of the primitive, such as line thickness, color, and style. Each attribute has an initial value, provided as a default setting. When you call an output function, the current values of its attributes are *bound* to the function, so that the output primitive reflects the current attribute values.

Output attribute functions can radically affect how the output primitive appears on the workstation surface. For instance, depending on the current text attribute values, the positioning point passed to the output function GKS$TEXT may be the center point for the text string, the position of the first character in the text string, or the position of the last character in the text string. The text output attributes also determine whether the string runs horizontal to the workstation X axis, vertical to the workstation X axis, or at a specified angle on the display surface.

This chapter requires that you be familiar with the following attribute issues:

- The types of attributes available for a primitive.
- The effects of using individual and bundled attributes.
- The use of nominal sizes and scale factors.
- The use of foreground and background color.

For complete information on these and any other output attribute topics, refer to Chapter 6, Output Attribute Functions.

5.3 Transformations and the DEC GKS Coordinate Systems

When you input and output primitives, you are actually working with three different coordinate systems. These coordinate systems are as follows:

- World coordinate system
- Normalized device coordinate (NDC) system
- Device coordinate system (workstation specific)

Notice that several program examples in this chapter generate a picture of an arrow on the workstation surface. When specifying points in the arrow, you use the world coordinate system. The programs pass the world coordinate points (0.1, 0.5), (0.9, 0.5), (0.7, 0.6), (0.7, 0.4), and (0.9, 0.5).

A world coordinate plane is an imaginary Cartesian coordinate plane, with a central point (0, 0), and an X and Y axis that extend to infinity in all directions. You establish the limits of the X and Y boundaries within which you want to work, and then create the picture you wish to output. All of the DEC GKS output functions accept the world coordinate points of the particular output primitive to be drawn. By default, the primitive must be drawn using the coordinate range ([0,1] x [0,1]). (All of the program examples in this chapter use this default coordinate range.)

DEC GKS use two separate transformations to translate your world coordinates to NDC coordinates, and to translate your NDC coordinates to device coordinates. During this process, portions of your primitives may be removed from the final picture due to clipping. You need to be aware of the effects of transformations and clipping on your generated output primitives. For complete information concerning transformations, refer to Chapter 7, Transformation Functions.

5.4 Output Deferral

When you output primitives, a workstation may postpone the generation of the image on the workstation surface depending on the workstation's capabilities. This postponement is called output *deferral.*

DEC GKS supports four deferral modes for its supported workstations. The deferral modes, in increasing order of deferral, are GKS$K_ASAP (generates output as soon as possible), GKS$K_BNIG (generates output before the next interaction globally), GKS$K_BNIL (generates output before the next interaction locally), and GKS$K_ASTI (at some time).

You can specify a suggested level of deferral by calling the function GKS$SET_DEFER_STATE. Depending on the capabilities of the workstation, it can defer output at the highest level up to the level specified in the call to GKS$SET_DEFER_STATE.

For detailed information concerning GKS$SET_DEFER_STATE and deferral, refer to Chapter 4, Control Functions.

5.5 Output Inquiries

The following list presents the inquiry functions that you can use to obtain output information when writing device-independent code:

GKS$INQ_AVAIL_GDP
GKS$INQ_ACTIVE_WS
GKS$INQ_GDP
GKS$INQ_PIXEL
GKS$INQ_PIXEL_ARRAY
GKS$INQ_PIXEL_ARRAY_DIM
GKS$INQ_OPERATING_STATE
GKS$INQ_TEXT_EXTENT

For more information concerning device-independent programming, refer to the *DEC GKS User Manual*.

5.6 DEC GKS Output Function Descriptions

This section describes the DEC GKS output functions in detail.

CELL ARRAY

Operating States: WSAC, SGOP

Description

The function GKS$CELL_ARRAY divides a designated rectangular area into cells, and displays each cell in a specific color or shade.

You pass a two-dimensional array containing color index values as one argument to this function. GKS$CELL_ARRAY maps the color index values to corresponding cells within a rectangular area of the workstation surface. In addition to the color index array, you specify an offset into the color array (a *starting element*), the number of array columns to be mapped, and the number of array rows to be mapped.

There is a one-to-one correspondence between the number of specified array columns and rows, and the number of columns and rows by which DEC GKS divides the cell array rectangle. Each of the columns within the rectangle is of equal width, and each of the rows within the rectangle is of equal height. DEC GKS maps the color index values from each specified color index array element to the corresponding cell, moving from the starting point towards the diagonal point along the X axis.

For more information concerning the initial color index values for a given workstation, refer to the *DEC GKS Device Specifics Reference Manual.* To alter the color associated with a certain index value, you can use the GKS function GKS$SET_COLOR_REP (refer to Chapter 6, Output Attribute Functions).

Syntax

GKS$CELL_ARRAY *(starting_point_x, starting_point_y, diagonal_point_x, diagonal_point_y, offset_column_number, offset_row_number, number_of_columns, number_of_rows, color_index_values)*

GCA *(spx, spy, dpx, dim_x, dim_y, scol, srow, ncols, nrows, cindex)*

gcellarray *(rectangle, dimensions, color)*

Arguments

starting_point_x
starting_point_y

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments designate any corner of the cell array rectangle as the cell array starting point. You pass these arguments as world coordinate values.

diagonal_point_x
diagonal_point_y

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments specify the corner of the cell array that is diagonal to the starting point, in world coordinates.

offset_column_number
offset_row_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

These arguments are the offset into the color index array. The offset determines the number of array columns and rows that you specify as arguments to

GKS$CELL_ARRAY. For instance, if the offset is the first element of the array, you can specify the full dimensions of the color index array as the "number of columns to map" and the "number of rows to map."

If you specify an offset that begins mapping in the interior of the color index array, you can only specify the remaining columns and rows as the number of columns and rows to map. For instance, if you pass an array three columns by four rows and specify an offset at element (2, 2), DEC GKS can only map the indexes from the two columns and three rows that follow the offset array element: elements (2, 2), (3, 2), (2, 3), (3, 3), (2, 4), (3, 4). If you attempt to divide the rectangle into more columns and rows than those from the offset to the last element in the array, DEC GKS generates an error.

Example 5–1 reproduces the situation described in the last paragraph.

number_of_columns
number_of_rows

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

These arguments specify into how many columns and rows DEC GKS divides the cell array rectangle. If you attempt to divide the rectangle into more columns and rows than those from the offset element to the last element of the color index array, DEC GKS generates an error. For more information, refer to the previous argument descriptions and to Example 5–1.

color_index_values

data type:	**2-D array (integer)**
access:	**read-only**
mechanism:	**by descriptor**

This argument is the two-dimensional array that contains the color index values. Previous arguments determine the offset element, the number of array columns to traverse, and the number of array rows to traverse.

When DEC GKS maps the values from the color index array to the cell array, it starts at the corner cell containing the cell array starting point and maps towards the diagonal point along the X axis. DEC GKS divides the cell array rectangle into the number of cells equal to the number of array elements specified by the previous arguments. Each cell is equal in width, and each cell is equal in height.

For more information on the color index array, refer to Example 5–1.

NOTE

The GKS$CELL_ARRAY example uses a FORTRAN column-major color index array. You may produce a different cell array if you use a language that supports row-major arrays (such as Pascal, C, and so forth).

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-33	DECGKS$_ERROR_NEG_33	Array descriptor is not acceptable in routine ****
5	GKS$_ERROR_5	GKS not in proper state: GKS shall be either in the state WSAC or in the state SGOP in routine ****
91	GKS$_ERROR_91	Dimensions of color array are invalid in routine ****

Program Example

Example 5–1 illustrates the use of the function GKS$CELL_ARRAY. Following the program example, Figures 5–3, 5–4, and 5–5 illustrate the program's effect on a VT241 workstation.

Example 5–1: Cell Array Output

```
C     This program displays three cell array rectangles.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER COLORS( 3, 4 ), NUM_COLS, NUM_ROWS, START_COL,
     * START_ROW, WS_ID
      REAL START_X, START_Y, DIAG_X, DIAG_Y
❶     DATA COLORS /3,2,0,  1,3,2,  0,2,0,  3,1,1/
      DATA WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      START_X = 0.1
      START_Y = 0.9
      DIAG_X  = 0.9
      DIAG_Y  = 0.1

      START_ROW = 1
      START_COL = 1
      NUM_COLS = 3
      NUM_ROWS = 4

❷     CALL GKS$CELL_ARRAY(START_X, START_Y, DIAG_X, DIAG_Y,
     * START_COL, START_ROW, NUM_COLS, NUM_ROWS, %DESCR( COLORS ))

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the screen.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

      START_X = 0.1
      START_Y = 0.9
      DIAG_X  = 0.9
      DIAG_Y  = 0.1

      START_ROW = 2
      START_COL = 2
      NUM_COLS = 2
      NUM_ROWS = 3
```

(continued on next page)

Example 5–1 (Cont.): Cell Array Output

```
❸       CALL GKS$CELL_ARRAY(START_X, START_Y, DIAG_X, DIAG_Y,
       * START_COL, START_ROW, NUM_COLS, NUM_ROWS, %DESCR( COLORS ))

C      Release deferred output. Pause.  Type RETURN when you are finished
C      viewing the screen.
       CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
       READ(5,*)

       START_X = 0.1
       START_Y = 0.1
       DIAG_X  = 0.9
       DIAG_Y  = 0.9

       START_ROW = 2
       START_COL = 2
       NUM_COLS = 2
       NUM_ROWS = 3

❹       CALL GKS$CELL_ARRAY(START_X, START_Y, DIAG_X, DIAG_Y,
       * START_COL, START_ROW, NUM_COLS, NUM_ROWS, %DESCR( COLORS ))
       CALL GKS$DEACTIVATE_WS( WS_ID )
       CALL GKS$CLOSE_WS( WS_ID )
       CALL GKS$CLOSE_GKS()
       END
```

The following numbers correspond to the numbers in the previous example:

❶ This code initializes the variable COLORS with three columns and four rows of color index values. By default, the index number 0 specifies black; the index number 1 specifies green; the index number 2 specifies red, and the index number 3 specifies blue. This program uses the same color index array as the one pictured in Figure 5–1.

❷ This code creates a rectangle whose starting corner point is (.1, .9) in world coordinates and whose diagonal corner point is (.9, .1) in world coordinates. DEC GKS divides the cell array rectangle into three columns of four rows.

GKS$CELL_ARRAY uses the color index value in the first column and in the first row of the array COLORS. GKS$CELL_ARRAY assigns one color to one cell until it colors all three columns and all four rows. After this call, the upper left cell array is blue, the next cell to the right is red, the last cell in that row is black, and so forth. To compare the contents of the

color index array with the cell array on the screen, compare Figure 5–1 with Figure 5–3.

You use the FORTRAN argument list built-in function %DESCR() to pass an array by descriptor.

❸ In this call to GKS$CELL_ARRAY, the offset array element is now (2, 2). This call divides the rectangle into the maximum allowable columns (2) and rows (3) given the location of the offset element. To compare the contents of the color index array with the cell array on the screen, compare Figure 5–2 with Figure 5–4.

❹ In this call to GKS$CELL_ARRAY, the starting corner point is the lower left corner. DEC GKS divides the cell along the X axis from the starting corner point cell to the diagonal point cell. Notice how the cell array appears to be a mirror image of the other cell array generated by this program. To compare the contents of the color index array with the cell array on the screen, compare Figure 5–2 with Figure 5–5.

Figure 5–1: The Maximum Number of Cells in the Cell Array

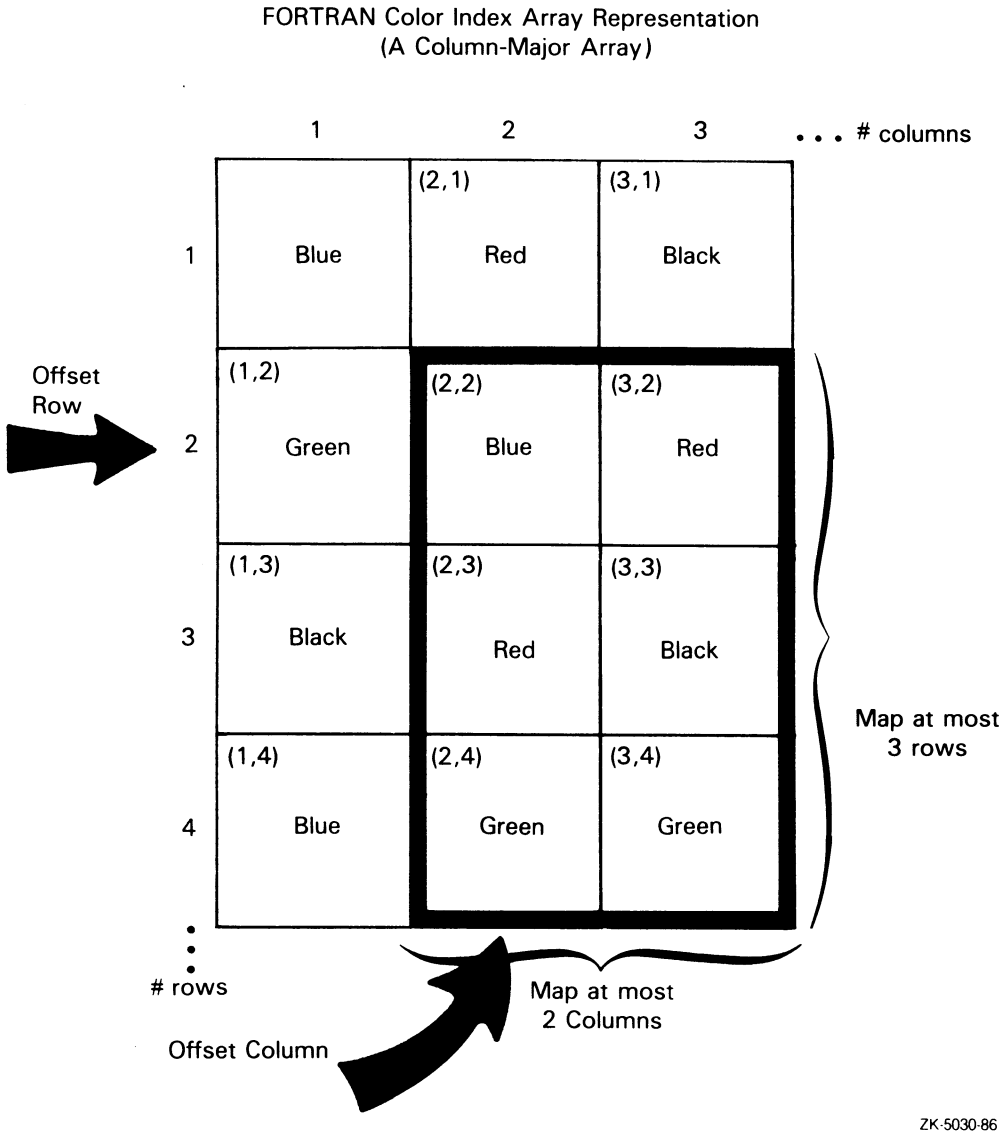
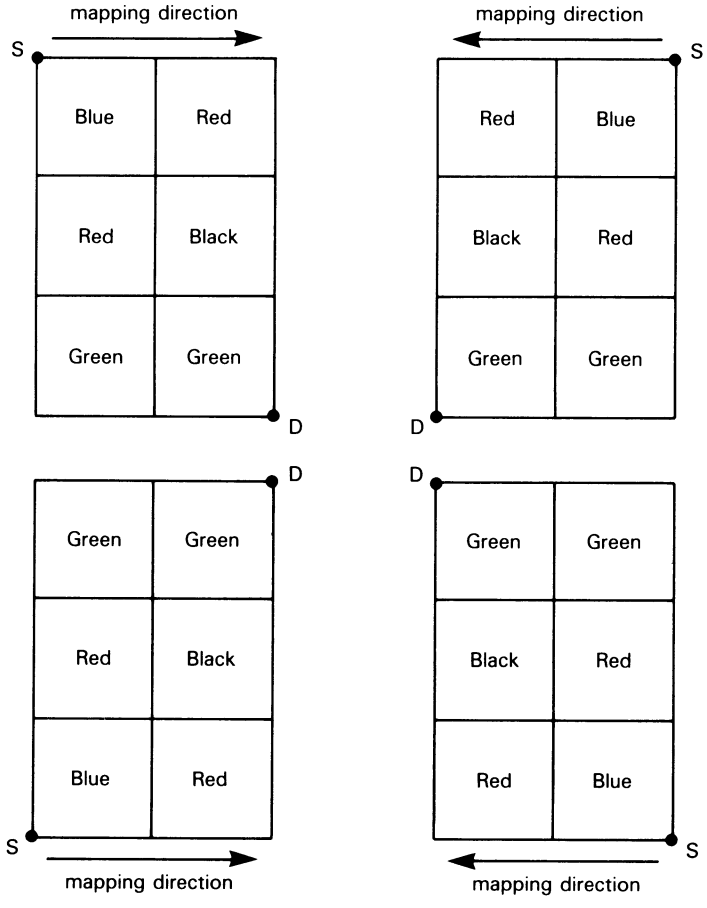


Figure 5–2: Possible Mapping Directions Using the Cell Array

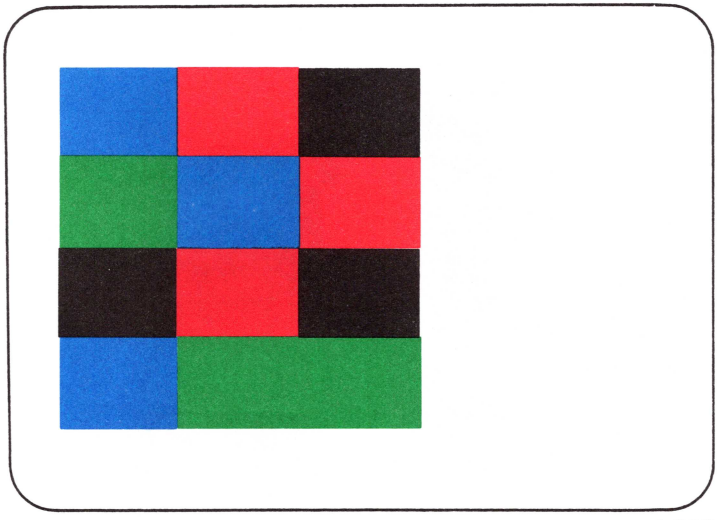
VAX GKS maps from the starting point toward the diagonal point, always along the X axis.



By changing the starting and diagonal points, you can choose between four different mirror images.

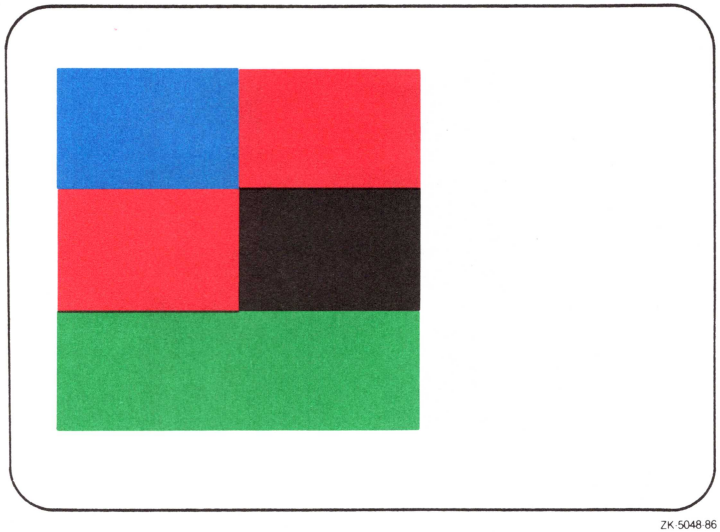
ZK-5163-86

Figure 5–3: Cell Array Output—VT241



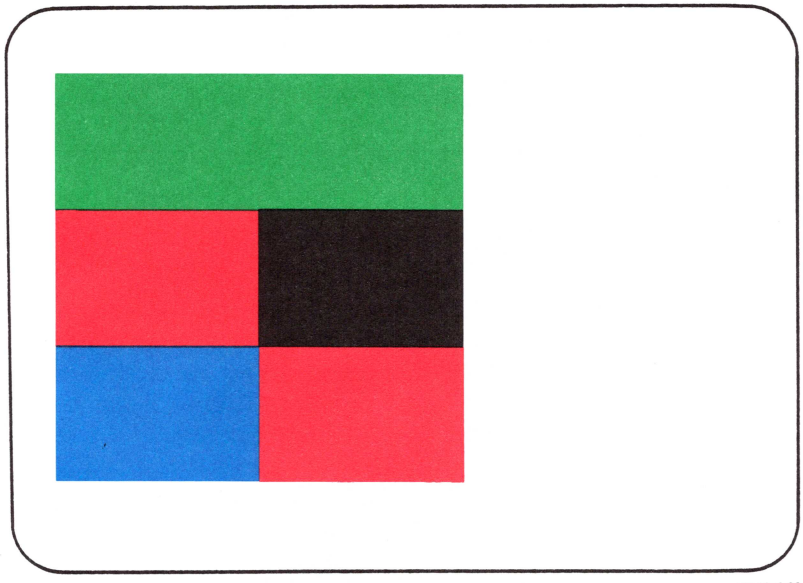
ZK-5047-86

Figure 5–4: The Second Call for Cell Array Output—VT241



ZK-5048-86

Figure 5–5: The Third Call for Cell Array Output—VT241



ZK-5049-86

FILL AREA

Operating States: WSAC, SGOP

Description

The function GKS$FILL_AREA draws a polygon and fills it with an interior style.

The fill area interior style can be either hollow, solid, hatched, or patterned. For instance, the default fill area interior style for most supported workstation types is hollow (GKS$FILL_AREA draws the outline of the polygon, leaving the interior hollow). For information on how to change the fill area attributes, refer to Chapter 6, Output Attribute Functions.

Syntax

GKS$FILL_AREA *(number_of_points, x_coordinates, y_coordinates)*

GFA *(number_of_points, px, py)*

gfillarea *(number_of_points, points)*

Arguments

number_of_points

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument specifies the number of points in the polygon.

x_coordinates
y_coordinates

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

These arguments are arrays containing the X and the Y values of the polygon's world coordinate points. The number of array elements should match the value of number_of_points.

You do not have to specify a closed polygon. If you do not specify a closed polygon, DEC GKS connects the last point specified to the first point.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
5	GKS$_ERROR_5	GKS not in proper state: GKS shall be either in the state WSAC or in the state SGOP in routine ****
100	GKS$_ERROR_100	Number of points is invalid in routine ****

Program Example

Example 5–2 illustrates the use of the function GKS$FILL_AREA. Following the program example, Figure 5–6 illustrates the program's effect on a VT241 workstation.

Example 5–2: Fill Area Output

```
C     This program splits a rectangle in half and then
C     fills both halves with different interior styles.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER NUM_POINTS, WS_ID, PURPLE_PATTERN
      REAL PX( 3 ), PY( 3 ), PX2( 3 ), PY2( 3 )

❶     DATA PX  /.1, .9, .1/
      DATA PY  /.1, .9, .9/
      DATA PX2 /.1, .9, .9/
      DATA PY2 /.1, .1, .9/
      DATA WS_ID / 1 /, NUM_POINTS / 3 /, PURPLE_PATTERN / 2 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )

❸     CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_PATTERN )
      CALL GKS$SET_FILL_STYLE_INDEX( PURPLE_PATTERN )
❹     CALL GKS$FILL_AREA( NUM_POINTS, PX2, PY2 )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ These arrays contain the polygon's X and Y world coordinates. For example, the first element of the array PX is the X value of the first point, and the first element of the array PY is the Y value. The first world coordinate point in the polygon is (.1, .1).

❷ In the call to GKS$FILL_AREA, you specify that there are three points in the polygon, as well as the arrays containing the world coordinate points.

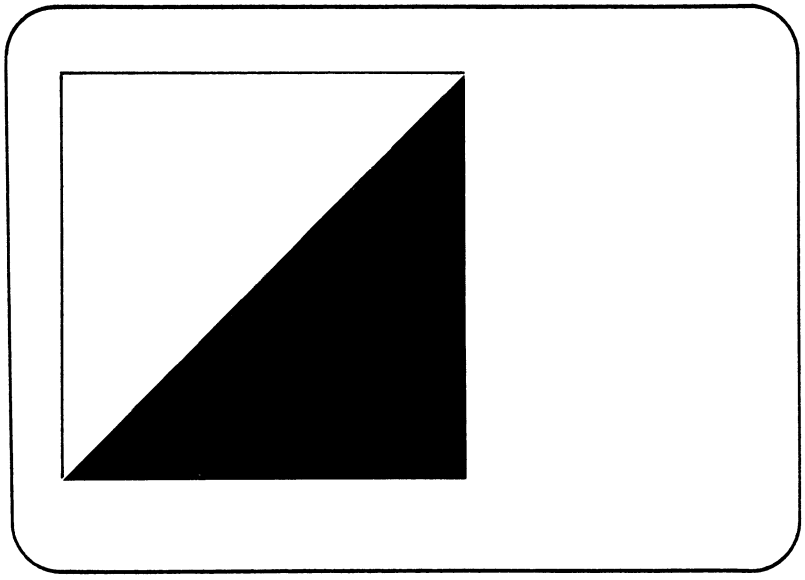
❸ This code changes the interior fill attribute from hollow to pattern, and then specifies the index number of a pattern. On the VT241, the default pattern for pattern index value 2 generates a purple pattern. If you are not working with a VT241, you may have to specify a different pattern index value to

achieve the same effects. For more information on GKS$SET_FILL_INT_STYLE and GKS$SET_FILL_STYLE_INDEX, refer to Chapter 6, Output Attribute Functions.

❹ This code fills the other triangle using the second set of X and Y world coordinate values.

Figure 5–6 shows the screen of a VT241 terminal after the program runs to completion.

Figure 5–6: Fill Area—VT241



ZK-5050-86

GDP

Operating States: WSAC, SGOP

Description

The function GKS$GDP generates a **generalized drawing primitive** (GDP) of the type you specify, using given points and any additional information contained in a data record. A GDP is a device-specific primitive that is not supported as a primitive by GKS. For instance, using DEC GKS, you can pass a center world coordinate point and a perimeter point to this function, and the specified workstation that supports such a GDP draws a circle on the workstation surface.

The definition of the particular GDP primitive specifies which sets of attributes the workstation uses to generate the primitive. For instance, the GDPs that generate circles use the set of polyline attributes.

Depending on the workstation-dependent requirements of the GDP, DEC GKS may or may not generate the primitive if certain points fall outside the current workstation window. If a workstation cannot generate a GDP because points fall outside of the current workstation window, DEC GKS generates an error message.

For complete descriptions of all the DEC GKS supported GDPs, refer to Appendix I, DEC GKS GDPs and Escapes.

Syntax

GKS$GDP *(number_of_points, x_coordinates, y_coordinates, gdp_id, data_record, data_record_size)*

GGDP *(number_of_points, px, py, gdp_id, dim_dr, dr)*

ggdp *(number_of_points, points, function, data)*

Arguments

number_of_points

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument specifies the number of points in the GDP.

x_coordinates
y_coordinates

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

These arguments are arrays containing the X and the Y values of the GDP's world coordinate points. The number of array elements should match the value of number_of_points.

gdp_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the GDP identifier. For a complete list of GDP identifiers, refer to Appendix I, DEC GKS GDPs and Escapes.

data_record

data type: **address (record)**
access: **read-only**
mechanism: **by reference**

This argument is the address of the GDP data record. For information concerning the structure of this data record, refer to Chapter 1, Introduction to DEC GKS. For information concerning data record structures required for specific GDPs, refer to Appendix I, DEC GKS GDPs and Escapes.

data_record_size

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the GDP data record size in bytes.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
5	GKS$_ERROR_5	GKS not in proper state: GKS shall be either in the state WSAC or in the state SGOP in routine ****
100	GKS$_ERROR_100	Number of points is invalid in routine ****
102	GKS$_ERROR_102	Generalized Drawing Primitive identifier is invalid in routine ****
103	GKS$_ERROR_103	Content of Generalized Drawing Primitive data record is invalid in routine ****
104	GKS$_ERROR_104	At least one active workstation is not able to generate the specified Generalized Drawing Primitive in routine ****
105	GKS$_ERROR_105	At least one active workstation is not able to generate the specified Generalized Drawing Primitive under the current transformation and clipping rectangle in routine ****

Program Example

Example 5–3 illustrates the use of the function GKS$GDP. Following the program example, Figure 5–7 illustrates the program's effect on a VT241 workstation.

Example 5–3: Generalized Drawing Primitive Output

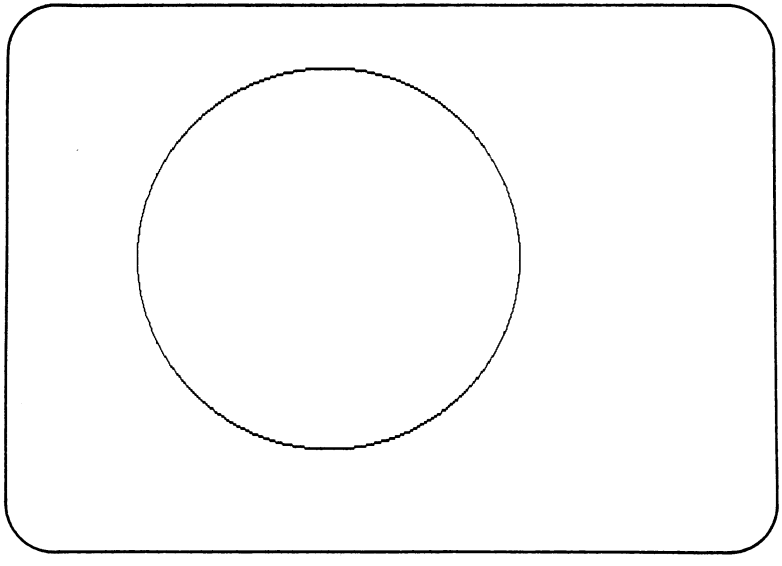
```
C     This program creates a filled circle using the GDP
C     GKS$K_GDP_CIRCLE_3PT.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, DATA_RECORD( 1 ), RECORD_SIZE

      REAL PX( 3 ), PY( 3 )

      DATA PX / 0.1, 0.5, 0.9 /
      DATA PY / 0.5, 0.1, 0.5 /
      DATA WS_ID / 1 /, NUM_POINTS / 3 /, RECORD_SIZE / 0 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT,
     * GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Pass the identifier of the filled circle.
      CALL GKS$GDP( NUM_POINTS, PX, PY,
(1)   * GKS$K_GDP_CIRCLE_3PT, DATA_RECORD, RECORD_SIZE )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

(1) The constant GKS$K_GDP_CIRCLE_3PT specifies the GDP identification number –102. This GDP creates a circle using three points on the circle's circumference. This particular GDP does not require a data record to perform its task. Notice that DEC GKS uses the current polyline attributes to create the circle. For more information concerning available GDPs, refer to Appendix I, DEC GKS GDPs and Escapes.

Figure 5–7 shows the screen of a VT241 terminal after the program runs to completion.

Figure 5–7: Generalized Drawing Primitive Output—VT241



ZK-5833-HC

POLYLINE

Operating States: WSAC, SGOP

Description

The function GKS$POLYLINE draws one or more straight lines, connecting the world coordinate points in the order specified.

By default, GKS$POLYLINE draws line segments as solid lines, at the nominal width, in the foreground color. For information on changing the polyline attributes, refer to Chapter 6, Output Attribute Functions.

Syntax

GKS$POLYLINE *(number_of_points, x_coordinates, y_coordinates)*

GPL *(number_of_points, px, py)*

gpolyline *(number_of_points, points)*

Arguments

number_of_points

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument specifies the number of points in the polyline.

x_coordinates
y_coordinates

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

These arguments are arrays containing the X and the Y values of the polyline's world coordinate points. The number of array elements should match the value of number_of_points.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
5	GKS$_ERROR_5	GKS not in proper state: GKS shall be either in the WSAC or in the state SGOP in routine ****
100	GKS$_ERROR_100	Number of points is invalid in routine ****

Program Example

Example 5–4 illustrates the use of the function GKS$POLYLINE. Following the program example, Figure 5–8 illustrates the program's effect on a VT241 workstation.

Example 5–4: Polyline Output

```
C     This program draws an arrow.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS
      REAL PX( 5 ), PY( 5 )

❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/
      DATA WS_ID / 1 /, NUM_POINTS / 5 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$POLYLINE( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

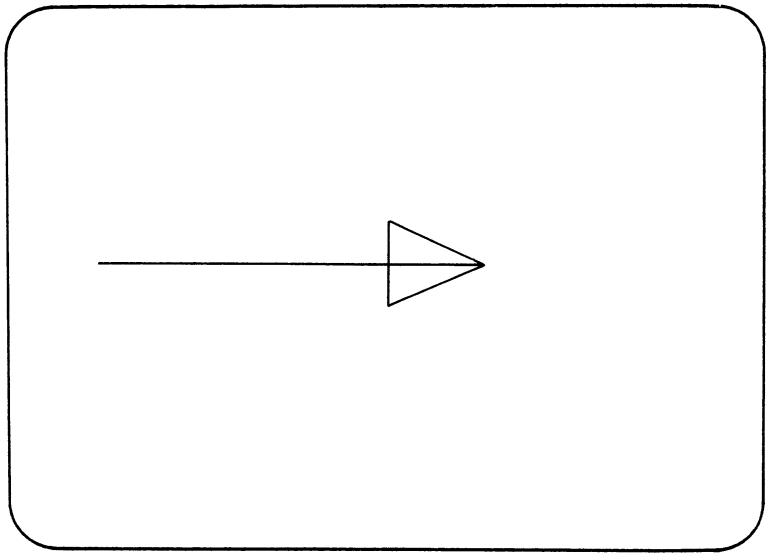
The following numbers correspond to the numbers in the previous example:

❶ These arrays contain the X and Y values of the polyline's world coordinates. For example, the first element of the array PX is the X value of the first point, and the first element of the array PY is the Y value of the first point. The first world coordinate point in the polyline is (.1, .5).

❷ In the call to GKS$POLYLINE, you specify the number of world coordinate points and the array variables.

Figure 5–8 shows the screen of a VT241 terminal after the program runs to completion.

Figure 5–8: Polyline Output—VT241



ZK-5051-86

POLYMARKER

Operating States: WSAC, SGOP

Description

The function GKS$POLYMARKER places one or more special symbols called **markers** at the specified world coordinates.

By default, GKS$POLYMARKER produces an asterisk marker, at the nominal size, in the workstation-specific default foreground color. For information on changing these polymarker attributes, refer to Chapter 6, Output Attribute Functions.

If clipping is enabled, and if the marker coordinate point is *outside* of the clipping rectangle, then DEC GKS clips the entire marker. If clipping is enabled, if the marker coordinate point is *inside* of the clipping rectangle, and if portions of the marker exceed the boundaries of the clipping rectangle, the extent of the clipping is device dependent. For more information concerning clipping, refer to Chapter 7, Transformation Functions.

Syntax

GKS$POLYMARKER *(number_of_points, x_coordinates, y_coordinates)*

GPM *(number_of_points, px, py)*

gpolymarker *(number_of_points, points)*

Arguments

number_of_points

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument specifies the number of marker coordinate locations.

x_coordinates
y_coordinates

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

These arguments are arrays containing the X and the Y values of the marker's world coordinate points. The number of array elements should match the value of number_of_points.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
5	GKS$_ERROR_5	GKS not in proper state: GKS shall be either in the state WSAC or in the state SGOP in routine ****
100	GKS$_ERROR_100	Number of points is invalid in routine ****

Program Example

Example 5–5 illustrates the use of the function GKS$POLYMARKER. Following the program example, Figure 5–9 illustrates the program's effect on a VT241 workstation.

Example 5–5: Polymarker Output

```
C     This program generates five markers.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS
      REAL PX( 5 ), PY( 5 )

❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/
      DATA WS_ID / 1 /, NUM_POINTS / 5 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$POLYMARKER( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

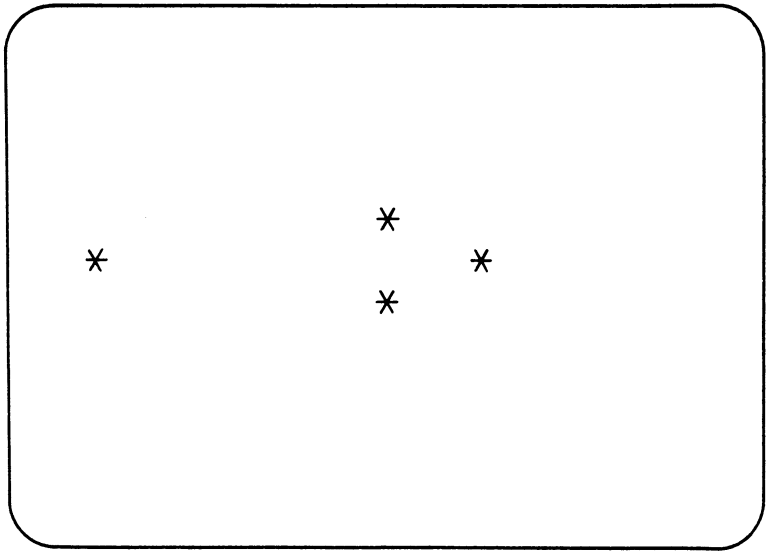
The following numbers correspond to the numbers in the previous example:

❶ These arrays contain the X and Y values of the markers' world coordinates. For example, the first element of the array PX is the X value of the first marker point, and the first element of the array PY is the Y value. The first marker world coordinate point is (.1, .5).

❷ In the call to GKS$POLYMARKER, you specify the number of world coordinate points to be marked and the array variables.

Figure 5–9 shows the screen of a VT241 terminal after the program runs to completion.

Figure 5–9: Polymarker Output—VT241



ZK-5052-86

TEXT

Operating States: WSAC, SGOP

Description

The function GKS$TEXT writes a character string that DEC GKS positions according to the specified world coordinates and according to the current text attributes. The shape of the characters within the text string may vary depending on the current text attributes, the current normalization transformation, and the particular workstation capabilities.

There are text attributes that control the nongeometric text properties (text font and precision, character expansion factor, character spacing, and text color index) and the geometric text properties (character height, character-up vector, character path, and character alignment). To determine the options concerning the appearance of the text string on the workstation surface, review the text attribute functions in Chapter 6, Output Attribute Functions.

The amount of the string that DEC GKS clips depends on both the current text attributes and the particular workstation capabilities. For string precision, DEC GKS clips the string in a workstation-dependent manner. For character precision, DEC GKS clips the string character by character. For stroke precision, DEC GKS clips the string exactly at the normalization viewport. For more information concerning clipping and normalization viewports, refer to Chapter 7, Transformation Functions.

Syntax

GKS$TEXT *(x_coordinate, y_coordinate, text_string)*

GTX *(px, py, text)*

GTXS - subset *(px, py, ltext, text)*

gtext *(position, text)*

Arguments

x_coordinate
y_coordinate

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments are the world coordinates that position the text string.

Depending on the current text attributes, DEC GKS positions the first character, the last character, or the middle of the text string at this world coordinate point. By default, DEC GKS positions the first character in the string at this point, and writes subsequent characters to the right of the starting point. You should review GKSSET_TEXT_UPVEC, GKSSET_TEXT_PATH, and GKS$SET_TEXT_ALIGN to see the options concerning placement of the text string on the workstation surface. For more information, refer to Chapter 6, Output Attribute Functions.

text_string

data type: **string**
access: **read-only**
mechanism: **by descriptor**

This argument is the ASCII text string to be written to the workstation surface.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-33	DECGKS$_ERROR_NEG_33	Array descriptor is not acceptable in routine ****
-34	DECGKS$_ERROR_NEG_34	String length less than or equal to 0 in routine ****

Error Number	Completion Status Code	Message
5	GKS$_ERROR_5	GKS not in proper state: GKS shall be either in the state WSAC or in the state SGOP in routine ****
101	GKS$_ERROR_101	Invalid code in string in routine ****

Program Example

Example 5–6 illustrates the use of the function, GKS$TEXT. Following the program example, Figure 5–10 illustrates the program's effect on a VT241 workstation.

Example 5–6: Text Output

```
C     This program writes a string to the workstation using the
C     default text attributes.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID
      REAL WORLD_X, WORLD_Y, LARGER
      DATA WS_ID / 1 /, WORLD_X / 0.1 /, WORLD_Y / 0.5 /,
     * LARGER / 0.04 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Increase text height by four times for clarity.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )

      CALL GKS$TEXT( WORLD_X, WORLD_Y,
     * 'TEXT: 0.04 WC units' )
```

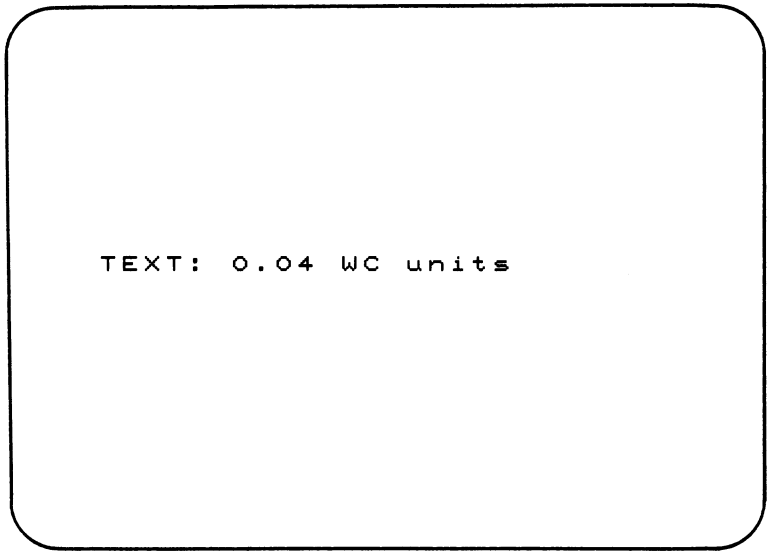
(continued on next page)

Example 5–6 (Cont.): Text Output

```
CALL GKS$DEACTIVATE_WS( WS_ID )
CALL GKS$CLOSE_WS( WS_ID )
CALL GKS$CLOSE_GKS()
END
```

Figure 5–10 shows the screen of a VT241 terminal after the program runs to completion.

Figure 5–10: Text Output—VT241



ZK-5053-86

Chapter 6

Output Attribute Functions

The DEC GKS output attribute functions affect the appearance of generated output primitives. The following list presents the output attribute functions by category:

Category	GKS Functions
Fill Area Attributes	GKS$SET_FILL_COLOR_INDEX, GKSSET_FILL_INDEX, GKSSET_FILL_INT_STYLE, GKS$SET_FILL_STYLE_INDEX, GKS$SET_PAT_REF_PT, GKS$SET_PAT_SIZE
Polyline Attributes	GKS$SET_PLINE_COLOR_INDEX, GKSSET_PLINE_INDEX, GKSSET_PLINE_LINETYPE, GKS$SET_PLINE_LINEWIDTH
Polymarker Attributes	GKS$SET_PMARK_COLOR_INDEX, GKSSET_PMARK_INDEX, GKSSET_PMARK_SIZE, GKS$SET_PMARK_TYPE
Text Attributes	GKSSET_TEXT_ALIGN, GKSSET_TEXT_COLOR_INDEX, GKSSET_TEXT_EXPFAC, GKSSET_TEXT_FONTPREC, GKSSET_TEXT_HEIGHT, GKSSET_TEXT_INDEX, GKSSET_TEXT_PATH, GKSSET_TEXT_SPACING, GKS$SET_TEXT_UPVEC
Aspect Source Flags	GKS$SET_ASF
Representations	GKSSET_COLOR_REP, GKSSET_FILL_REP, GKSSET_PAT_REP, GKSSET_PLINE_REP, GKSSET_PMARK_REP, GKSSET_TEXT_REP

The DEC GKS state list stores the current value of the output attributes for each output function. These attributes specify the exact appearance of the object drawn. For example, when you call GKS$POLYLINE, the attributes line type, width scale factor, and color specify the form, thickness, and color of the line. In the DEC GKS state list, these current attributes are stored in the entries *current line type, current line width scale factor,* and *current polyline color index.*

When you call a DEC GKS output function, the attributes are bound to the primitive. If the primitive's attributes are *individual*, then you cannot change these attributes; changes to output attributes only affect subsequent output. If the primitive's attributes are *bundled*, then you may be able to change the attributes of previously generated primitives by calling one of the representation functions, depending on the capabilities of your device. See Section 6.2 for more information concerning individual and bundled attributes.

6.1 Types of Attributes

Attributes can affect **geometric**, **nongeometric**, and **pick identification** aspects of a graphic image. The geometric and nongeometric aspects of a graphic image directly affect how the primitive appears on the workstation surface. The pick identification is used when performing pick input. For complete details concerning pick input, refer to Chapter 8, Input Functions.

Most output functions have nongeometric attributes that are changeable (cell arrays and generalized drawing primitives do not). Nongeometric attributes affect the style and the pattern of the output primitives (such as polyline color, text spacing, and fill area interior style). Since the nongeometric attributes are scale factors and *nominal* sizes, the effects of these attributes are device dependent.

Nominal sizes are the default sizes of markers and line widths as defined by a graphics handler. In most cases the nominal size is also the smallest size that a workstation can produce, but not always. DEC GKS multiplies the scale factor values by the nominal size to reset a marker size or polyline width. The default value for a scale factor is 1.0 (the nominal size multiplied by the value 1.0, producing no change in size).

Geometric attributes affect the size or positioning of text and fill area primitives (such as text height, character path, and pattern size). Fill area and text are the only two output primitives that have changeable geometric attributes. The geometric attributes are specified in world coordinate units. Therefore, since the world coordinates are device independent, the geometric attributes are device independent.

Table 6–1 lists the output attributes and whether an attribute is geometric or nongeometric.

Table 6–1: Geometric and Nongeometric Output Attributes

Function	Attribute	Type
Polyline	Polyline index	Nongeometric
	Line type	Nongeometric
	Line width scale factor	Nongeometric
	Polyline color index	Nongeometric
Polymarker	Polymarker index	Nongeometric
	Marker type	Nongeometric
	Marker size	Nongeometric
	Polymarker color index	Nongeometric
Text	Text index	Nongeometric
	Text height	Geometric
	Character up vector	Geometric
	Text path	Geometric
	Text alignment	Geometric
	Text font and precision	Nongeometric
	Character expansion factor	Nongeometric
	Character spacing	Nongeometric
	Text color index	Nongeometric
Fill area	Fill area index	Nongeometric
	Pattern size	Geometric
	Pattern reference point	Geometric
	Fill area interior style	Nongeometric
	Fill area style index	Nongeometric
	Fill area color index	Nongeometric

Notice that there are no attribute functions specifically designed to alter the cell array or the generalized drawing primitives (GDPs). A cell array is simply an array of indexes that point to the workstation's color table.

The GDP has no attributes specifically designed for it. Depending on the workstation-specific GDP data record, you may need to specify any number of the polyline, polymarker, text, or fill area attribute values, depending on the nature of the GDP.

6.2 Individual and Bundled Attribute Values

As stated previously, the current values of each attribute are listed individually in the DEC GKS state list. By default, a call to an output function uses these individual, attribute values to generate the primitive. Since DEC GKS stores these individual attributes in the DEC GKS state list, they are essentially device independent. If you specify attributes individually, you cannot change a primitive's appearance on the workstation surface once you generate it.

However, there is a second method used to specify attribute values. Each workstation can define a number of attribute *bundles* for an output primitive. Each bundle is an entry in a table that contains attribute values for each of the nongeometric values of that particular output primitive. DEC GKS stores bundle tables in the workstation state lists, thereby making the bundle table entries device dependent. You specify bundle table entries by specifying a bundle index value that points into the table. Most workstations provide a fill area bundle index 1, but the resulting fill area can look different on each workstation.

For example, a polyline bundle contains table entries for *polyline index, line type, line width scale factor,* and *polyline color index.* A workstation can define a bundle table entry with the index 1 that specifies a solid line type. The same workstation can define another bundle table entry with the index 2 that specifies a dashed line type. The output attributes associated with a bundle table index constitute that index's *representation.*

When you call an output function, DEC GKS uses the current, individual output values stored in the DEC GKS state list, by default. If you wish to use the device-dependent bundle table indexes, you must change the attribute's aspect source flag (ASF). The ASFs are described in Section 6.2.1.

If you use bundled attributes for primitives, you may be able to change the appearance of the generated primitive by redefining its bundle index representation. For many workstations, changing index representations requires an implicit regeneration, which erases all primitives not contained in segments. For complete information concerning the representation functions, refer to Section 6.2.2.

To review the initial individual output attributes, refer to Appendix C, DEC GKS Attribute Values. To review the bundle tables available on a given workstation, refer to the *DEC GKS Device Specifics Reference Manual.*

6.2.1 Aspect Source Flags (ASFs)

When you call an output function, DEC GKS uses the individual output attributes by default. To use bundle tables of attributes, you must establish a set of *aspect source flags* (ASFs).

The set of ASFs is a thirteen-element integer array, one element for every nongeometric output attribute (element 1 corresponds to the line type attribute, element 2 corresponds to the line width scale factor attribute, and so forth). Each element contains either the value GKS$K_ASF_BUNDLED (0) or the value GKS$K_ASF_INDIVIDUAL (1). By passing this array to the function GKS$SET_ASF, DEC GKS uses either the individual attribute value or the bundled value in the bundle table specified by the current bundle index.

For example, the following code illustrates the use of ASFs:

```
      .
      .
      .
      CALL GKS$SET_PLINE_LINETYPE( GKS$K_LINETYPE_SOLID )

C     This function call produces a solid line.
      CALL GKS$POLYLINE( 2, PX, PY )
      .
      .
      .
C     ASF_FLAGS is a thirteen-element integer array.  The first
      element corresponds to the line type attribute.
      ASF_FLAGS( 1 ) = GKS$K_ASF_BUNDLED
      CALL GKS$SET_ASF( ASF_FLAGS )
      .
      .
      .
C     Polyline bundle index number 3 specifies a dashed line
C     on this workstation.
      CALL GKS$SET_PLINE_INDEX( 3 )

C     This call produces a dashed line.
      CALL GKS$POLYLINE( 2, PX, PY )
      .
      .
      .
```

For a complete discussion of ASFs, refer to the GKS$SET_ASF function description in this chapter.

NOTE

If you store primitives in a segment and if you want to be able to change the primitive's appearance elsewhere in the program, you must set the primitive's ASF to be GKS$K_ASF_BUNDLED before you generate the primitive. In this way, the primitive's ASF is stored in the segment with the primitive. If you want to change the primitive's appearance, you call the appropriate SET REPRESENTATION function (see Section 6.2.2) for the primitive's bundle index. If you store the primitive in a segment using individual attributes, the appearance of the primitive cannot be changed.

6.2.2 Dynamic Changes and Implicit Regeneration

When working with bundled attributes, you can use any bundle index value predefined by your workstation. You can even alter the existing bundles table entries, or create new entries, using the representation functions (GKSSET_PLINE_REP, GKSSET_PMARK_REP, and so forth).

If you use the representation functions, use caution. Depending on the capabilities of your workstation, DEC GKS may implement the change immediately, or the change may require an implicit regeneration of the surface. An implicit regeneration clears the screen and only redraws the visible segments. You lose all primitives not contained in segments. Many of the DEC GKS supported workstations suppress implicit regenerations since they cause you to lose all primitives not contained in segments.

For a detailed discussion of implicit regeneration, refer to Chapter 4, Control Functions.

6.3 Foreground and Background Colors

The default color index value is 1, which corresponds to the workstation's *foreground color.* All of the default individual color indexes in the DEC GKS state list are set to the value 1.

On a GKS$K_WSCAT_OUTIN or GKS$K_WSCAT_OUTPUT workstation, the color of a "blank" surface is called the background color. The color of characters written to the workstation surface is called the foreground color.

Unless you change these color index values using the function GKS$SET_COLOR_REP, the color index value 0 corresponds to the workstation's default background color, and the color index value 1 corresponds to the workstation's default foreground color. If the workstation supports more than two color indexes, values greater than 1 correspond to alternative foreground colors.

6.4 Output Attribute Inquiries

The following list presents the inquiry functions that you can use to obtain output attribute information when writing device-independent code:

GKS$INQ_COLOR_FAC
GKS$INQ_COLOR_INDEXES
GKS$INQ_COLOR_REP
GKS$INQ_FILL_FAC
GKS$INQ_FILL_INDEXES
GKS$INQ_FILL_REP
GKS$INQ_INDIV_ATTB
GKS$INQ_MAX_WS_STATE_TABLE
GKS$INQ_OPEN_WS
GKS$INQ_PAT_FAC
GKS$INQ_PAT_INDEXES
GKS$INQ_PAT_REP
GKS$INQ_PLINE_FAC
GKS$INQ_PLINE_INDEXES
GKS$INQ_PLINE_REP
GKS$INQ_PMARK_FAC
GKS$INQ_PMARK_INDEXES
GKS$INQ_PMARK_REP
GKS$INQ_PREDEF_COLOR_REP
GKS$INQ_PREDEF_FILL_REP
GKS$INQ_PREDEF_PAT_REP
GKS$INQ_PREDEF_PLINE_REP
GKS$INQ_PREDEF_PMARK_REP
GKS$INQ_PREDEF_TEXT_REP
GKS$INQ_PRIM_ATTB
GKS$INQ_TEXT_FAC
GKS$INQ_TEXT_INDEXES
GKS$INQ_REP

For more information concerning device-independent programming, refer to the *DEC GKS User Manual*.

6.5 Function Descriptions

These sections describe each of the DEC GKS attribute functions by category: polyline attributes, polymarker attributes, text attributes, fill area attributes, attribute source flags, and representation functions.

Fill Area Attributes

The DEC GKS functions described in this section affect the following geometric and nongeometric fill attributes:

- Color index (nongeometric)
- Bundle index (nongeometric)
- Interior style (nongeometric)
- Style index (nongeometric)
- Pattern reference point (geometric)
- Pattern size (geometric)

Each of these functions can alter the default values used in subsequent calls to the GKS$FILL_AREA function. For more information concerning GKS$FILL_AREA, refer to Chapter 5, Output Functions.

SET FILL AREA COLOR INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_FILL_COLOR_INDEX sets the *current fill color index* entry in the DEC GKS state list to the specified index value.

Syntax

GKS$SET_FILL_COLOR_INDEX *(color_index)*

GSFACI *(cindex)*

gsetfillcolourind *(colour)*

Arguments

color_index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the fill area color index. The default value for the fill area color index entry is the value 1, which designates the default foreground color. If a device cannot fill an area with the specified color, DEC GKS uses workstation-dependent color. For more information concerning predefined color indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
92	GKS$_ERROR_92	Color index is less than zero in routine ****

Program Example

Example 6–1 illustrates the use of the function GKS$SET_FILL_COLOR_INDEX. Following the program example, Figure 6–1 illustrates the program's effect on a VT241 workstation.

Example 6–1: Changing the Fill Color Index

```
C     This program changes the fill color of a triangle from
C     the default color green to the color blue.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, BLUE, NUM_POINTS
      DATA WS_ID / 1 /, BLUE / 3 /, NUM_POINTS / 3 /
❶     DATA PX  /.1, .9, .1/
      DATA PY  /.1, .9, .9/
      REAL PX( 3 ), PY( 3 )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$SET_FILL_COLOR_INDEX( BLUE )
      CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

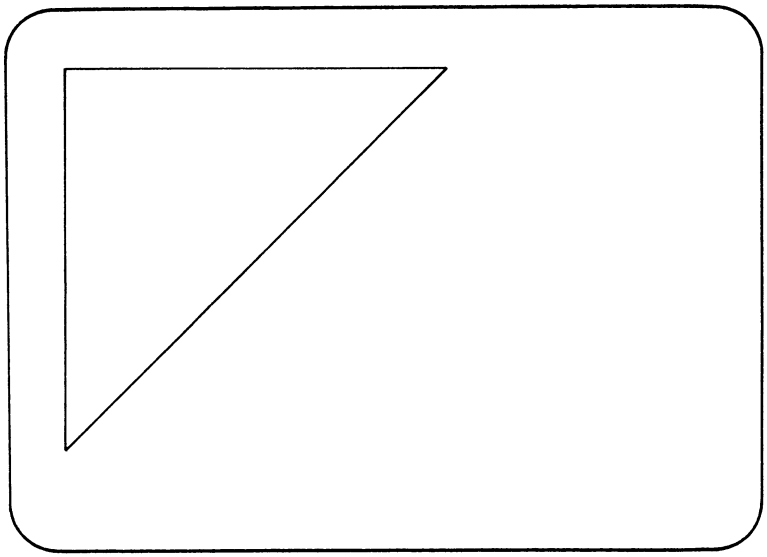
❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .1).

❷ The function GKS$SET_FILL_COLOR_INDEX changes the color from the default color to blue, but only if the fill area color ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Workstations other than the VT241 may predefine a different representation of color index 3 (a color other than blue).

Figure 6–1 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–1: Changing the Fill Color Index—VT241



ZK-5071-86

SET FILL AREA INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_FILL_INDEX establishes the index value pointing into the fill area bundle table. The fill area bundle table contains entries for the fill area interior style, fill area style index, and fill area color index attribute values. When calling GKS$FILL_AREA, DEC GKS uses the bundle table only if the corresponding attribute source flag has been set to GKS$K_ASF_BUNDLED.

For a list of the predefined fill area bundles for each workstation, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_FILL_INDEX *(index)*

GSFAI *(index)*

gsetfillind *(index)*

Arguments

index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the fill area bundle index. The default bundle index is the value 1. For more information concerning predefined fill area bundle indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
80	GKS$_ERROR_80	Fill area index is invalid in routine ****

Program Example

Example 6–2 illustrates the use of the function GKS$SET_FILL_INDEX. Following the program example, Figure 6–2 illustrates the program's effect on a VT241 workstation.

Example 6–2: Changing the Fill Index

```
C     This program sets the Attribute Source Flags (ASFs) to bundled,
C     and then displays the effects of using the first 10 index values
C     in calls to GKS$SET_FILL_INDEX.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, INCR
      DATA WS_ID / 1 /, NUM_POINTS / 3 /
❶     DATA PX  /.1, .9, .1/
      DATA PY  /.1, .9, .9/
      REAL PX( 3 ), PY( 3 )
      INTEGER FLAGS( 13 )
      CHARACTER*2 STR

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     FLAGS( 11 ) = GKS$K_ASF_BUNDLED
      FLAGS( 12 ) = GKS$K_ASF_BUNDLED
      FLAGS( 13 ) = GKS$K_ASF_BUNDLED
      CALL GKS$SET_ASF( FLAGS )

❸     DO 200 INCR = 1, 10, 1
      CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_ALWAYS )
      CALL GKS$SET_FILL_INDEX( INCR )
      CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )
❹     CALL LIB$CVT_DX_DX( %DESCR( INCR ), %DESCR( STR ) )
      CALL GKS$SET_TEXT_HEIGHT( 0.03 )
      CALL GKS$TEXT( .5, .4, 'Index:  ')
      CALL GKS$TEXT( .8, .4, %DESCR( STR ) )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

200   CONTINUE

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example.

❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .1).

❷ This code initializes the elements of the array that affect all of the nongeometric polymarker attributes. This code sets the fill area ASFs to GKS$K_ASF_BUNDLED. FLAGS(11) corresponds to the current fill area interior style; FLAGS(12) corresponds to the current fill area style index; and, FLAGS(13) corresponds to the fill area color index.

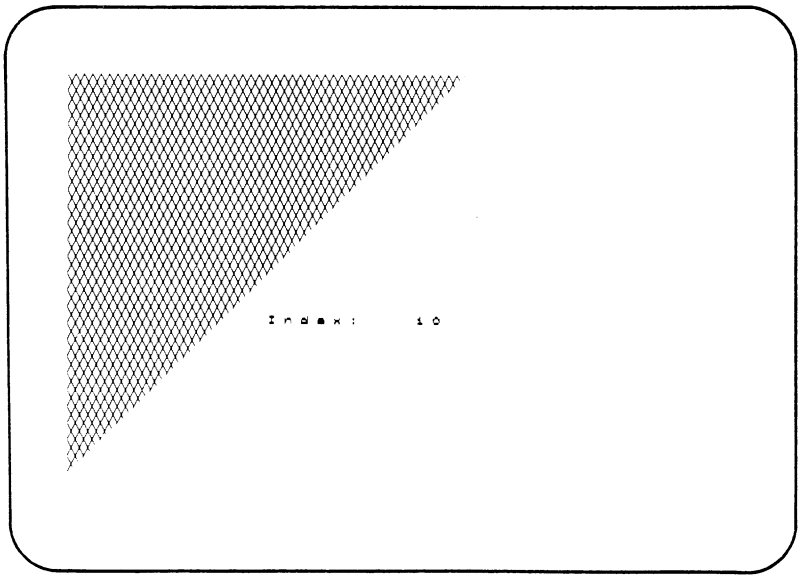
See the GKS$SET_ASF function description in this chapter for more information.

❸ This code displays the triangle using ten of the fill area index values available on the VT241. This code writes the index value that produced the fill area, to the right of the triangle.

❹ This VMS Run-Time Library Routine translates the variable INCR to a text string so that GKS$TEXT can write the fill area index value to the screen.

Figure 6–2 shows the screen of a VT241 terminal after the program has run to completion. The color of the filled area is green.

Figure 6–2: Changing the Fill Index—VT241



ZK-5072-86

SET FILL AREA INTERIOR STYLE

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_FILL_INT_STYLE sets the *current fill area interior style* entry in the DEC GKS state list to be hollow, solid, pattern, or hatched.

If you select solid, GKS$FILL_AREA fills the color designated by the current fill area color index.

If you select pattern, GKS$FILL_AREA replicates a pattern (alternating colors) to fill the interior of the polygon. The fill area attributes pattern size and pattern reference point define the size and position of the start of the pattern (see GKS$SET_PAT_SIZE and GKS$SET_PAT_REF_PT in this section). The fill area style index specifies the pattern to replicate (see GKS$SET_FILL_STYLE_INDEX in this section). Patterns cover underlying primitives.

If you select hatched, GKS$FILL_AREA fills the interior of the polygon with a series of designs in the color specified by the fill area color index. The fill area style index specifies the chosen hatch style. White spaces within hatches do not cover underlying primitives.

Syntax

GKS$SET_FILL_INT_STYLE *(int_style)*

GSFAIS *(style)*

gsetfillintstyle *(style)*

Arguments

int_style

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument designates the fill area interior style. The argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_INTSTYLE_HOLLOW	Use an outline.
1	GKS$K_INTSTYLE_SOLID	Use a color.
2	GKS$K_INTSTYLE_PATTERN	Use a pattern.
3	GKS$K_INTSTYLE_HATCH	Use crossed or parallel lines.

Error Messages

Error Number	Completion Status Code	Message
-11	DECGKS$_ERROR_NEG_11	Invalid value specified for fill area interior style in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****

Program Example

Example 6–3 illustrates the use of the function GKS$SET_FILL_INT_STYLE. Following the program example, Figure 6–3 illustrates the program's effect on a VT241 workstation.

Example 6–3: Changing the Fill Area Interior Style

```
C     This program splits a rectangle in half and then
C     fills both halves with different styles.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, DARK_BLUE_PAT
      DATA WS_ID / 1 /, NUM_POINTS / 3 /,
     * DARK_BLUE_PAT / 6 /
❶     DATA PX  /.1, .9, .1/
      DATA PY  /.1, .9, .9/
      DATA PX2 /.1, .9, .9/
      DATA PY2 /.1, .1, .9/
      REAL PX( 3 ), PY( 3 ), PX2( 3 ), PY2( 3 )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )
❸     CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_PATTERN )
❹     CALL GKS$SET_FILL_STYLE_INDEX( DARK_BLUE_PAT )
      CALL GKS$FILL_AREA( NUM_POINTS, PX2, PY2 )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The arrays PX and PY contain the polygon's world coordinate values. For example, the first element in both arrays specifies the first point (.1, .1).

❷ In the call to GKS$FILL_AREA, you specify that there are three points in the polygon, as well as the arrays containing the world coordinate points.

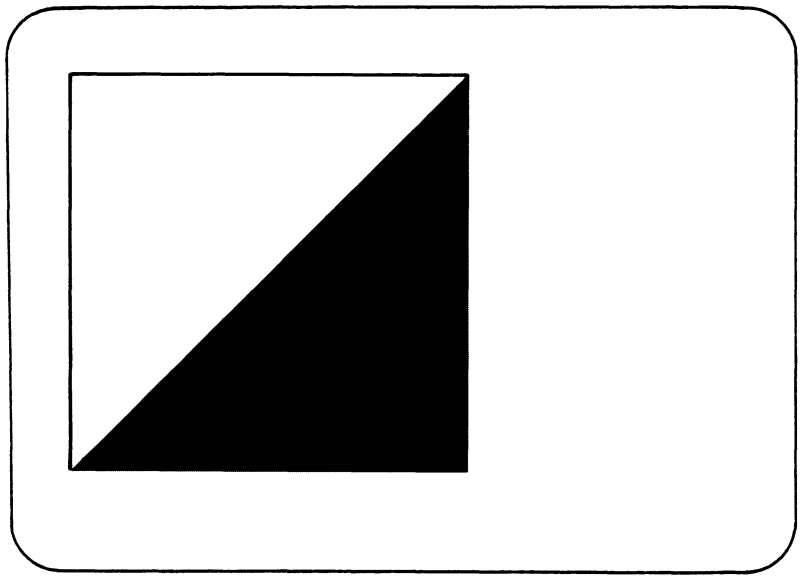
❸ This code changes the interior fill attribute from hollow to pattern as long as the current interior fill style ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

❹ This code changes the style to a style index that produces a dark blue pattern by alternating blue and black colors, as long as the current fill area style ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Workstations other than the VT241 may use other style values to specify a similar pattern. See GKS$SET_FILL_INT_STYLE in this section for more information.

Figure 6–3 shows the effects of the calls to GKS$FILL_AREA. The color of the top triangle is hollow green and the bottom triangle is a dark blue pattern.

Figure 6–3: Changing the Fill Area Interior Style—VT241



ZK-5073-86

SET FILL AREA STYLE INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_FILL_STYLE_INDEX sets the *current fill area style index* entry in the DEC GKS state list to the specified index value.

This function sets a specific pattern or hatch style to fill the interior of a polygonal fill area. If the interior style is hollow or solid, the current style index is ignored for the call to GKS$FILL_AREA. If the interior style is pattern, you must pass a pattern index value to this function. If the interior style is hatch, you must pass a hatch style value to this function. Since hatch styles are device dependent, the hatch style index is always a negative number.

Syntax

GKS$SET_FILL_STYLE_INDEX *(style_index)*

GSFASI *(sindex)*

gsetfillstyleind *(index)*

Arguments

style_index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the fill area style index. For information on the predefined hatch and pattern styles, refer to the *DEC GKS Device Specifics Reference Manual*. The initial fill area style index entry is the value 1. If you request a style index that is not available on a particular workstation, that workstation uses the style index 1. If the style index 1 is not present on the workstation, the result is workstation dependent.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
84	GKS$_ERROR_84	Style (pattern or hatch) index is equal to zero in routine ****

Program Example

Refer to Example 6–3 in this section for a program example using a call to GKS$SET_FILL_STYLE_INDEX.

SET PATTERN REFERENCE POINT

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PAT_REF_PT sets the geometric attribute *current pattern reference point* entry in the DEC GKS state list. This attribute represents the starting point for a pattern used to fill the designated area. DEC GKS uses this value for all subsequent calls to GKS$FILL_AREA until you specify another value.

NOTE

Most of the DEC GKS supported workstations do not fully support this function. Those workstations that do not, do accept the function call but do not make any changes to the pattern. For more information, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PAT_REF_PT *(x_coordinate, y_coordinate)*

GSPARF *(px, py)*

gsetpatrefpt *(patref)*

Arguments

x_coordinate
y_coordinate

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

These arguments designate the X and Y world coordinate unit values of the pattern starting point.

Error Messages

Error Number	Completion Status Code	Message
–20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****

SET PATTERN SIZE

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PAT_SIZE specifies the geometric attribute *current pattern size* entry in the DEC GKS state list, which is the height and width vectors in world coordinate units. The pattern size is replicated for use within a fill area. DEC GKS uses this value for all subsequent calls to GKS$FILL_AREA until you specify another value.

NOTE

Most of the DEC GKS supported workstations do not fully support this function. Those workstations that do not, do accept the function call but do not make any changes to the pattern. For more information, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PAT_SIZE *(pattern_width, pattern_height)*

GSPA *(px, py)*

gsetpatsize *(patsize)*

Arguments

pattern_width
pattern_height

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

These arguments specify the width and height in world coordinates units. DEC GKS begins replicating the pattern representation at the pattern reference point,

and continues until the polygonal fill area in world coordinate space is filled with the pattern.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
87	GKS$_ERROR_87	Pattern size value is not positive in routine ****

Polyline Attributes

The DEC GKS functions described in this section affect the following polyline attributes:

- Color index (nongeometric)
- Bundle index (nongeometric)
- Line type (nongeometric)
- Line width scale factor (nongeometric)

Depending on the ASF for the output attribute, each of these functions can alter the default values used in subsequent calls to GKS$POLYLINE. For more information concerning GKS$POLYLINE, refer to Chapter 5, Output Functions.

SET POLYLINE COLOR INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PLINE_COLOR_INDEX sets the *current polyline color index* DEC GKS state list entry to the specified index value.

Syntax

GKS$SET_PLINE_COLOR_INDEX *(color_index)*

GSPLCI *(cindex)*

gsetlinecolourind *(colour)*

Arguments

color_index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the polyline color index. DEC GKS uses the default foreground color for default polyline color index (1). For more information concerning predefined color indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
92	GKS$_ERROR_92	Color index is less than zero in routine ****

Program Example

Example 6–4 illustrates the use of the function GKS$SET_PLINE_COLOR_INDEX. Following the program example, Figure 6–4 illustrates the program's effect on a VT241 workstation.

Example 6–4: Changing the Polyline Color Index

```
C     This program changes the color of an arrow from green to blue.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, BLUE, NUM_POINTS
      DATA WS_ID / 1 /, BLUE / 3 /, NUM_POINTS / 5 /
      REAL PX( 5 ), PY( 5 )
❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$SET_PLINE_COLOR_INDEX( BLUE )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

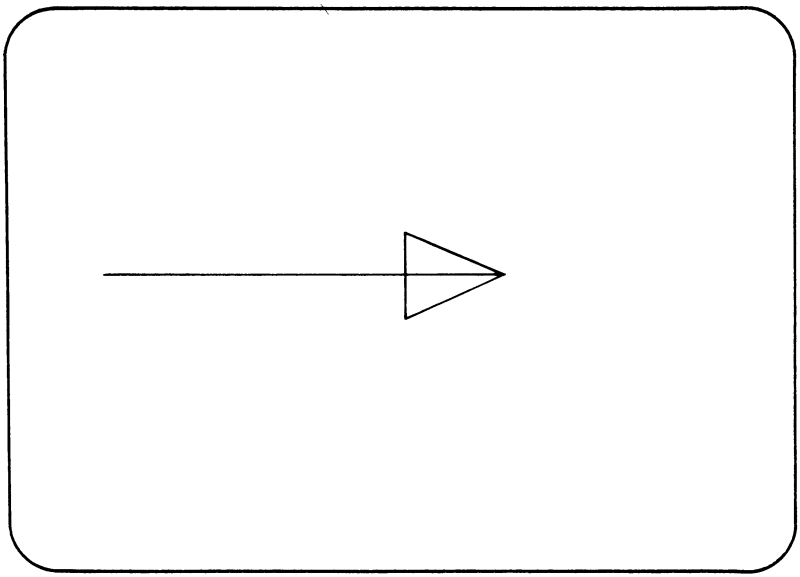
❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ The function GKS$SET_PLINE_COLOR_INDEX changes the color from the default color to blue, but only if the polyline color ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

The VT241 predefines the index value 3 to represent the color blue. Workstations other than the VT241 may predefine a different representation of color index 3 (a color other than blue).

Figure 6–4 shows the screen of a VT241 terminal after the program has run to completion. The arrow changed from the default color green to the color blue.

Figure 6–4: Changing the Polyline Color Index—VT241



ZK-5054-86

SET POLYLINE INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PLINE_INDEX establishes the index value pointing into the polyline bundle table. The polyline bundle table contains entries for the polyline color index, polyline line type, and polyline linewidth scale factor attribute values. When calling GKS$POLYLINE, DEC GKS uses the bundle table only if the corresponding attribute source flag has been set to GKS$K_ASF_BUNDLED.

For a list of the predefined polyline bundle entries for each workstation, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PLINE_INDEX *(index)*

GSPLI *(pindex)*

gsetlineind *(index)*

Arguments

index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the polyline bundle index. The default bundle index is the value 1. For more information concerning possible polyline bundle indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
60	GKS$_ERROR_60	Polyline index is invalid in routine ****

Program Example

Example 6–5 illustrates the use of the function GKS$SET_PLINE_INDEX. Following the program example, Figure 6–5 illustrates the program's effect on a VT241 workstation.

Example 6–5: Changing the Polyline Index

```
C     This program sets the Attribute Source Flags (ASFs) to bundled,
C     and then displays the effects of using the first 10 index values
C     in calls to GKS$SET_PLINE_INDEX.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, INCR
      DATA WS_ID / 1 /, NUM_POINTS / 5 /
      REAL PX( 5 ), PY( 5 )
❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/
      INTEGER FLAGS( 13 )
      CHARACTER*2 STR

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     FLAGS( 1 ) = GKS$K_ASF_BUNDLED
      FLAGS( 2 ) = GKS$K_ASF_BUNDLED
      FLAGS( 3 ) = GKS$K_ASF_BUNDLED
      CALL GKS$SET_ASF( FLAGS )

❸     DO 200 INCR = 1, 10, 1
      CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_ALWAYS )
      CALL GKS$SET_PLINE_INDEX( INCR )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
❹     CALL LIB$CVT_DX_DX( %DESCR( INCR ), %DESCR( STR ) )
      CALL GKS$SET_TEXT_HEIGHT( 0.03 )
      CALL GKS$TEXT( .1, .3, 'Index:  ')
      CALL GKS$TEXT( .4, .3, %DESCR( STR ) )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

200   CONTINUE

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example.

❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ This code initializes the elements of the array that affect all of the non-geometric polyline attributes. This code sets the three polyline ASFs to GKS$K_ASF_BUNDLED. FLAGS(1) corresponds to the line type; FLAGS(2) corresponds to the line width scale factor; and, FLAGS(3) corresponds to the polyline color index.

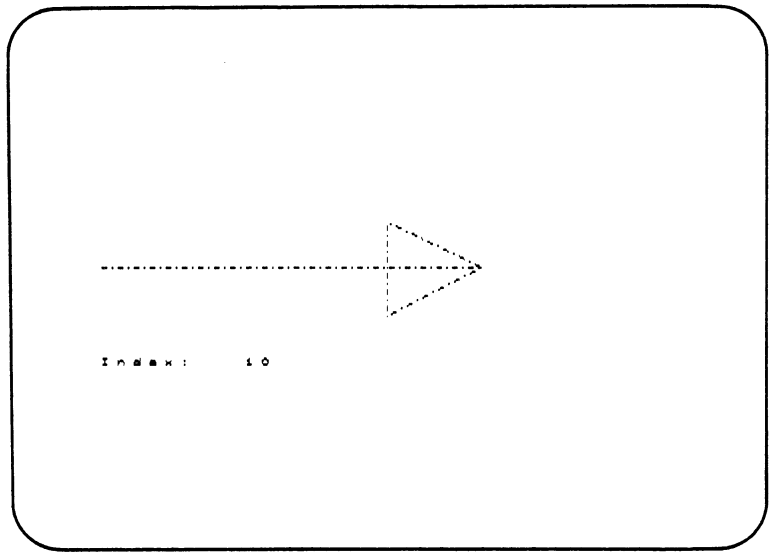
See the GKS$SET_ASF function description in this chapter for more information.

❸ This code displays the arrow using ten of the polyline index values available on the VT241. This code writes the index value that produced the polyline, in the lower left portion of the screen.

❹ This VMS Run-Time Library Routine translates the variable INCR to a text string so that GKS$TEXT can write the polyline index value to the screen.

Figure 6–5 shows the screen of a VT241 terminal after the program has run to completion. The color of the arrow is green.

Figure 6–5: Changing the Polyline Index—VT241



ZK-5055-86

SET LINETYPE

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PLINE_LINETYPE sets the *current polyline line type* entry in the DEC GKS state list to be solid, dashed, dotted, dashed-dotted, or any one of the device-dependent types.

Every workstation capable of output (DEC GKS category GKS$K_WSCAT_OUTPUT or GKS$K_WSCAT_OUTIN) defines at least four line types. For more information concerning possible polyline line type values, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PLINE_LINETYPE *(line_type)*

GSLN *(ltype)*

gsetlinetype *(type)*

Arguments

line_type

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the polyline line type. The argument can be any of the following values or constants.

Value	Constant	Description
< 0		Device-dependent types.
1	GKS$K_LINETYPE_SOLID	Use solid line.
2	GKS$K_LINETYPE_DASHED	Use dashed line.
3	GKS$K_LINETYPE_DOTTED	Use dotted line.
4	GKS$K_LINETYPE_DASHED_DOTTED	Use dashed-dotted line.
> = 5		Reserved: future standardization.

The default for the current line type value is 1, which displays a solid line. If you specify an unsupported line type, DEC GKS uses GKS$K_LINETYPE_SOLID. For more information concerning predefined line type indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
63	GKS$_ERROR_63	Specified linetype is equal to zero in routine ****

Program Example

Example 6–6 illustrates the use of the function GKS$SET_PLINE_LINETYPE. Following the program example, Figure 6–6 illustrates the program's effect on a VT241 workstation.

Example 6–6: Changing the Polyline Line Type

```
C     This program changes the solid lines of an arrow to
C     dashed and dotted lines.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS
      DATA WS_ID / 1 /, NUM_POINTS / 5 /
      REAL PX( 5 ), PY( 5 )
❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$SET_PLINE_LINETYPE( GKS$K_LINETYPE_DASHED_DOTTED )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

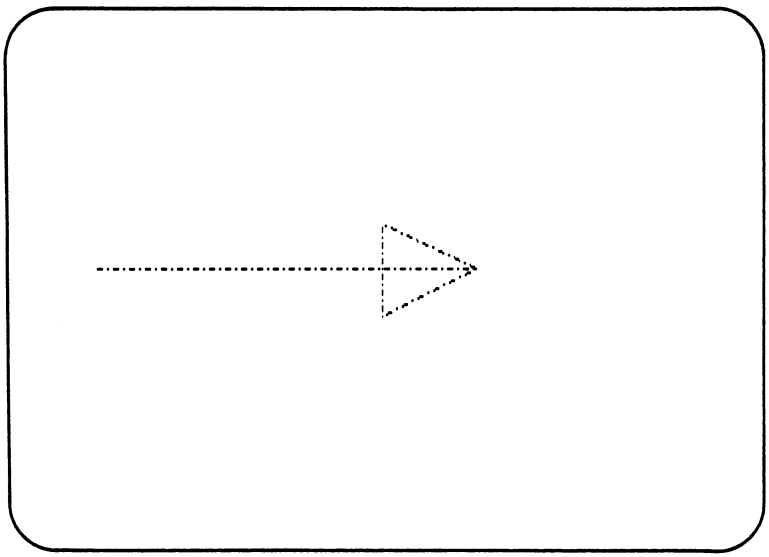
The following numbers correspond to the numbers in the previous example:

❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ The function GKS$SET_PLINE_LINETYPE changes the line type from the default type to dashed and dotted, only if the line type ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Figure 6–6 shows the screen of a VT241 terminal after the program has run to completion. The arrow changed from the default line type solid to the type dashed-dotted.

Figure 6–6: Changing the Polyline Line Type—VT241



ZK-5056-86

SET LINEWIDTH SCALE FACTOR

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PLINE_LINEWIDTH sets the *current polyline line width scale factor* entry in the DEC GKS state list.

DEC GKS calculates line width as the nominal line width, multiplied by the line width scale factor. The line width scale factor is a real number that you pass to GKS$SET_PLINE_LINEWIDTH. The graphics handler maps the value to the nearest available line width defined by the graphics handler.

Syntax

GKS$SET_PLINE_LINEWIDTH *(line_width_ scale_factor)*

GSLWSC *(lwidth)*

gsetlinewidth *(width)*

Arguments

line_width_scale_factor

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

This argument is the line width scale factor. The default for the current entry is the value 1.0, which outputs a line of the nominal width.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
65	GKS$_ERROR_65	Linewidth scale factor is less than zero in routine ****

Program Example

Example 6–7 illustrates the use of the function GKS$SET_PLINE_LINEWIDTH. Following the program example, Figure 6–7 illustrates the program's effect on a VT241 workstation.

Example 6–7: Changing the Polyline Line Width

```
C     This program increases the line width of an arrow 5 times.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS
      REAL MIN_TIMES_FIVE
      DATA WS_ID / 1 /, NUM_POINTS / 5 /,
     * MIN_TIMES_FIVE / 5.0 /
      REAL PX( 5 ), PY( 5 )
❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$SET_PLINE_LINEWIDTH( MIN_TIMES_FIVE )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

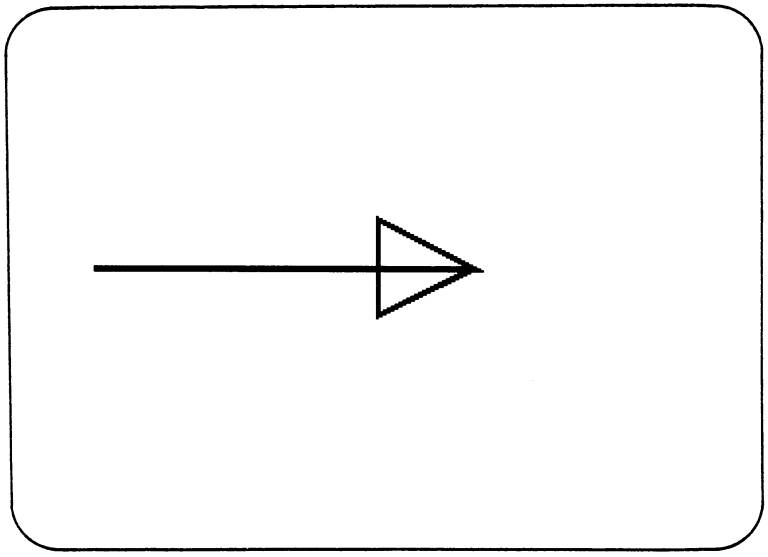
The following numbers correspond to the numbers in the previous example:

❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ The function GKS$SET_PLINE_LINEWIDTH changes the line width from the nominal width to five times the nominal width, only if the line width scale factor ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Figure 6–7 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–7: Changing the Polyline Line Width—VT241



ZK-5057-86

Polymarker Attributes

The DEC GKS functions described in this section affect the following polymarker attributes:

- Color index (nongeometric)
- Bundle index (nongeometric)
- Line type (nongeometric)
- Size (nongeometric)

Each of these functions can alter the default values used in subsequent calls to the GKS$POLYMARKER function. For more information concerning GKS$POLYMARKER, refer to Chapter 5, Output Functions.

SET POLYMARKER COLOR INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PMARK_COLOR_INDEX sets the *current polymarker color index* entry in the DEC GKS state list to the specified value.

Syntax

GKS$SET_PMARK_COLOR_INDEX *(color_index)*

GSPMCI *(cindex)*

gsetmarkercolourind *(colour)*

Arguments

color_index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the polymarker color index. The default value for the polymarker color index entry is the value 1, which designates the workstation's default foreground color. For more information concerning predefined color index values, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
92	GKS$_ERROR_92	Color index is less than zero in routine ****

Program Example

Example 6–8 illustrates the use of the function GKS$SET_PMARK_COLOR_INDEX. Following the program example, Figure 6–8 illustrates the program's effect on a VT241 workstation.

Example 6–8: Changing the Polymarker Color Index

```
C     This outputs five blue asterisks
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, BLUE, NUM_POINTS
      DATA WS_ID / 1 /, BLUE / 3 /, NUM_POINTS / 5 /
      REAL PX( 5 ), PY( 5 )
❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$SET_PMARK_COLOR_INDEX( BLUE )
      CALL GKS$POLYMARKER( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

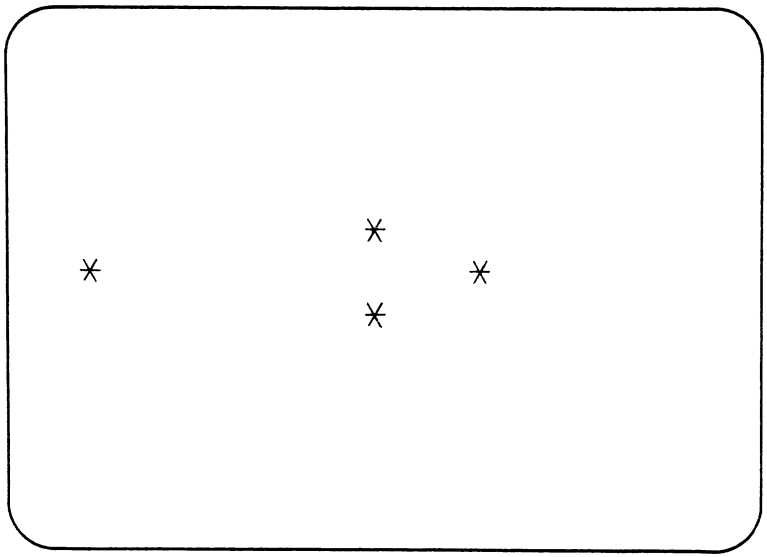
❶ PX contains the markers' X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ The function GKS$SET_PMARK_COLOR_INDEX changes the color from the default color to blue, but only if the polymarker color ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Workstations other than the VT241 may predefine a different representation of color index 3 (a color other than blue).

Figure 6–8 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–8: Changing the Polymarker Color Index—VT241



ZK-5058-86

SET POLYMARKER INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PMARK_INDEX establishes the index value pointing into the polymarker bundle table. The polymarker bundle table contains entries for the polymarker color index, polymarker type, and polymarker size scale factor attribute values. When calling GKS$POLYMARKER, DEC GKS uses the bundle table only if the corresponding attribute source flag has been set to GKS$K_ASF_BUNDLED.

For a list of the predefined polymarker area bundles for each workstation, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PMARK_INDEX *(index)*

GSPMI *(pindex)*

gsetmarkerind *(index)*

Arguments

index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the polymarker bundle index. The default bundle index is the value 1. For more information concerning predefined polymarker bundle table indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
66	GKS$_ERROR_66	Polymarker index is invalid in routine ****

Program Example

Example 6–9 illustrates the use of the function GKS$SET_PMARK_INDEX. Following the program example, Figure 6–9 illustrates the program's effect on a VT241 workstation.

Example 6–9: Changing the Polymarker Index

```
C     This program sets the Attribute Source Flags (ASFs) to bundled,
C     and then displays the effects of using the first 10 index values
C     in calls to GKS$SET_PMARK_INDEX.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, INCR
      DATA WS_ID / 1 /, NUM_POINTS / 5 /
      REAL PX( 5 ), PY( 5 )
❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/
      INTEGER FLAGS( 13 )
      CHARACTER*2 STR

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     FLAGS( 4 ) = GKS$K_ASF_BUNDLED
      FLAGS( 5 ) = GKS$K_ASF_BUNDLED
      FLAGS( 6 ) = GKS$K_ASF_BUNDLED
      CALL GKS$SET_ASF( FLAGS )

❸     DO 200 INCR = 1, 10, 1
      CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_ALWAYS )
      CALL GKS$SET_PMARK_INDEX( INCR )
      CALL GKS$POLYMARKER( NUM_POINTS, PX, PY )
❹     CALL LIB$CVT_DX_DX( %DESCR( INCR ), %DESCR( STR ) )
      CALL GKS$SET_TEXT_HEIGHT( 0.03 )
      CALL GKS$TEXT( .1, .3, 'Index:  ')
      CALL GKS$TEXT( .4, .3, %DESCR( STR ) )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

200   CONTINUE

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example.

❶ PX contains the markers' X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ This code initializes the elements of the array that affect all of the non-geometric polymarker attributes. This code sets the polymarker ASFs to GKS$K_ASF_BUNDLED. FLAGS(4) corresponds to the marker type; FLAGS(5) corresponds to the marker size scale factor; and, FLAGS(6) corresponds to the polymarker color index.

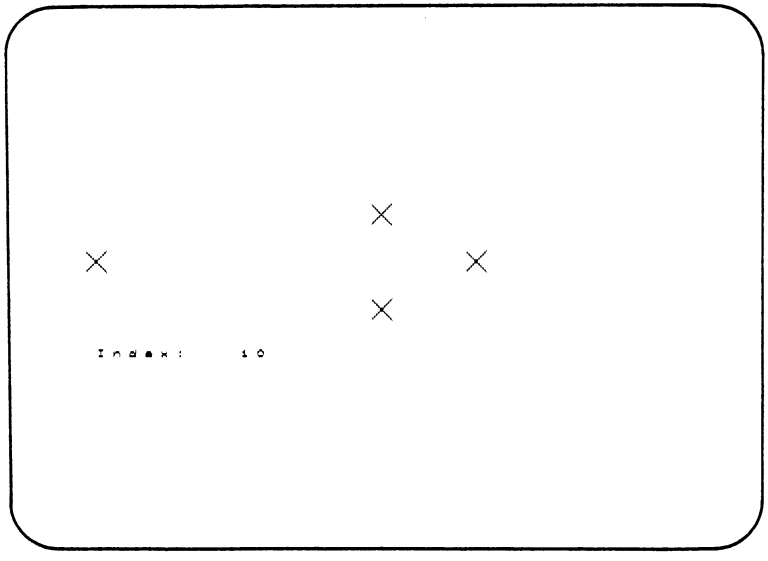
See the GKS$SET_ASF function description in this chapter for more information.

❸ This code displays the arrow using ten of the polymarker index values available on the VT241. This code writes the index value that produced the polymarker, in the lower left portion of the screen.

❹ This VMS Run-Time Library Routine translates the variable INCR to a text string so that GKS$TEXT can write the fill area index value to the screen.

Figure 6–9 shows the screen of a VT241 terminal after the program has run to completion. The markers are green.

Figure 6–9: Changing the Polymarker Index—VT241



ZK-5059-86

SET MARKER TYPE

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PMARK_TYPE sets the *current marker type* entry in the DEC GKS state list to be dots, plus signs, asterisks, circles, diagonal crosses, or any of the device-dependent types.

Every workstation capable of output (of DEC GKS category GKS$K_WSCAT_OUTPUT or GKS$K_WSCAT_OUTIN) defines at least five polymarker types. For more information concerning predefined polymarker type, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PMARK_TYPE *(marker_type)*

GSMK *(mtype)*

gsetmarkertype *(type)*

Arguments

marker_type

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the polymarker type. The argument can be any of the following values or constants.

Value	Constant	Description
< 0		Device-dependent types.
1	GKS$K_MARKERTYPE_DOT	Use dots (.).
2	GKS$K_MARKERTYPE_PLUS	Use plus signs (+).
3	GKS$K_MARKERTYPE_ASTERISK	Use asterisks (*).
4	GKS$K_MARKERTYPE_CIRCLE	Use circles (o).
5	GKS$K_MARKERTYPE_DIAGONAL_CROSS	Use diagonal crosses (X).
> = 6		Reserved: future standardization.

The default index for the current polymarker type entry is GKS$K_MARKERTYPE_ASTERISK. For more information concerning possible line type indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
69	GKS$_ERROR_69	Specified marker type is equal to zero in routine ****

Program Example

Example 6–10 illustrates the use of the function GKS$SET_PMARK_TYPE. Following the program example, Figure 6–10 illustrates the program's effect on a VT241 workstation.

Example 6–10: Changing the Polymarker Type

```
C     This program draws five circles.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS
      DATA WS_ID / 1 /, NUM_POINTS / 5 /
      REAL PX( 5 ), PY( 5 )
❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$SET_PMARK_TYPE( GKS$K_MARKERTYPE_CIRCLE )
      CALL GKS$POLYMARKER( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

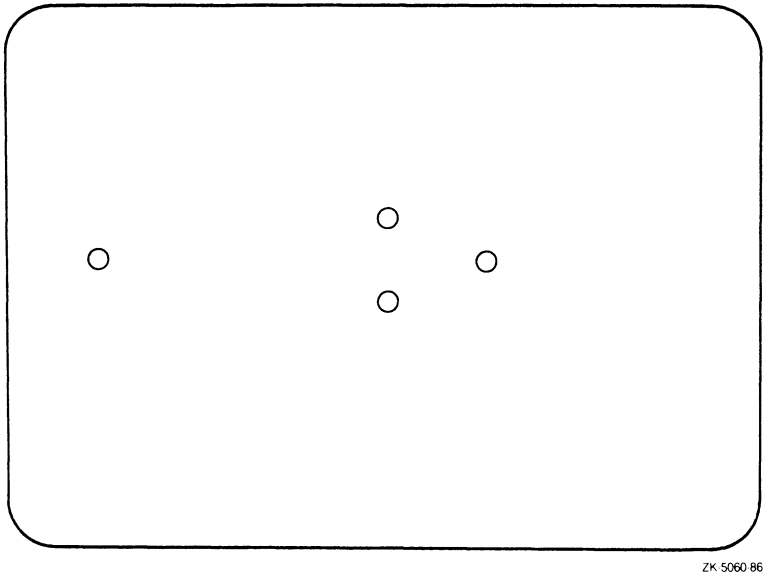
The following numbers correspond to the numbers in the previous example:

❶ PX contains the markers' X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ The function GKS$SET_PMARK_TYPE changes the polymarker type from the default type to circles, only if the marker type ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Figure 6–10 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–10: Changing the Polymarker Marker Type—VT241



ZK-5060-86

SET MARKER SIZE SCALE FACTOR

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_PMARK_SIZE sets the *current marker size scale factor* entry in the DEC GKS state list to the specified value for all polymarker types.

DEC GKS calculates marker size for all types (except the dot marker type) as the nominal marker size multiplied by the marker size scale factor. The marker size scale factor is a real number that you pass to GKS$SET_PMARK_SIZE. The graphics handler maps the value to the nearest available marker size defined by the handler. (The dot marker type is always the smallest dot that the workstation can produce.)

Syntax

GKS$SET_PMARK_SIZE *(marker_size_scale_factor)*

GSMKSC *(sfactor)*

gsetmarkersize *(size)*

Arguments

marker_size_scale_factor

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

This argument is the marker size scale factor. The default for the current entry is the value 1.0, which outputs a marker of the nominal size as defined by the graphics handler.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
71	GKS$_ERROR_71	Marker size scale factor is less than zero in routine ****

Program Example

Example 6–11 illustrates the use of the function GKS$SET_PMARK_SIZE. Following the program example, Figure 6–11 illustrates the program's effect on a VT241 workstation.

Example 6–11: Changing the Polymarker Size

```
C     This program draws five at five times
C     that of the default size.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS
      REAL MIN_TIMES_FIVE
      DATA WS_ID / 1 /, NUM_POINTS / 5 /,
     * MIN_TIMES_FIVE / 5.0 /
      REAL PX( 5 ), PY( 5 )
❶     DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$SET_PMARK_SIZE( MIN_TIMES_FIVE )
      CALL GKS$POLYMARKER( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

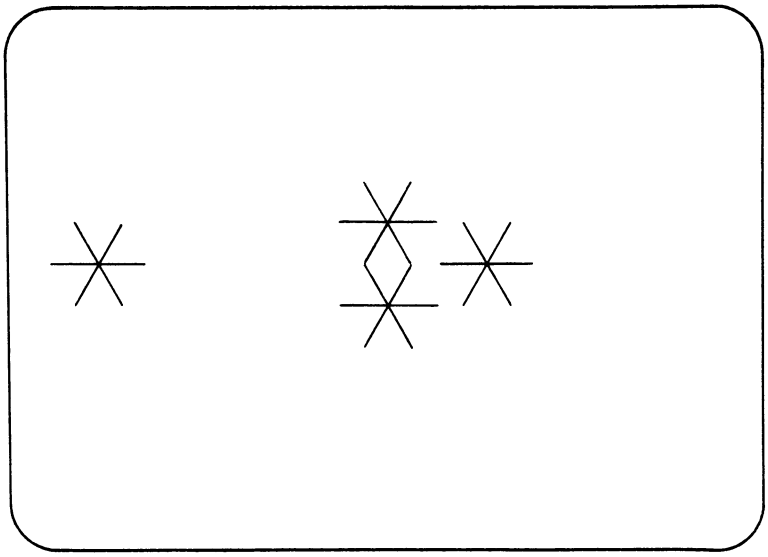
The following numbers correspond to the numbers in the previous example:

❶ PX contains the markers' X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ The function GKS$SET_PMARK_SIZE changes the marker size from the nominal width to five times the nominal width, only if the marker size scale factor ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Figure 6–11 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–11: Changing the Polymarker Size—VT241



ZK-5061-86

Text Attributes

The DEC GKS functions described in this section affect the following geometric and nongeometric text attributes:

- Alignment (geometric)
- Color (nongeometric)
- Expansion factor (nongeometric)
- Font and precision (nongeometric)
- Height (geometric)
- Bundle Index (nongeometric)
- Path (geometric)
- Spacing (nongeometric)
- Up vector (geometric)

Character strings are defined within a **text extent rectangle**. A text extent rectangle is an imaginary parallelogram that completely contains the character string to be written. The character string itself and the text attributes character height, character expansion factor, and character spacing define the limits of the text extent rectangle.

Each of these functions can alter the default values used in subsequent calls to the GKS$TEXT function. For more information concerning GKS$TEXT, refer to Chapter 5, Output Functions.

SET TEXT ALIGNMENT

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_TEXT_ALIGN sets the *current text alignment* entry in the DEC GKS state list to a value that specifies the positioning of the text extent rectangle. DEC GKS uses this value for all subsequent calls to GKS$TEXT until you specify another value.

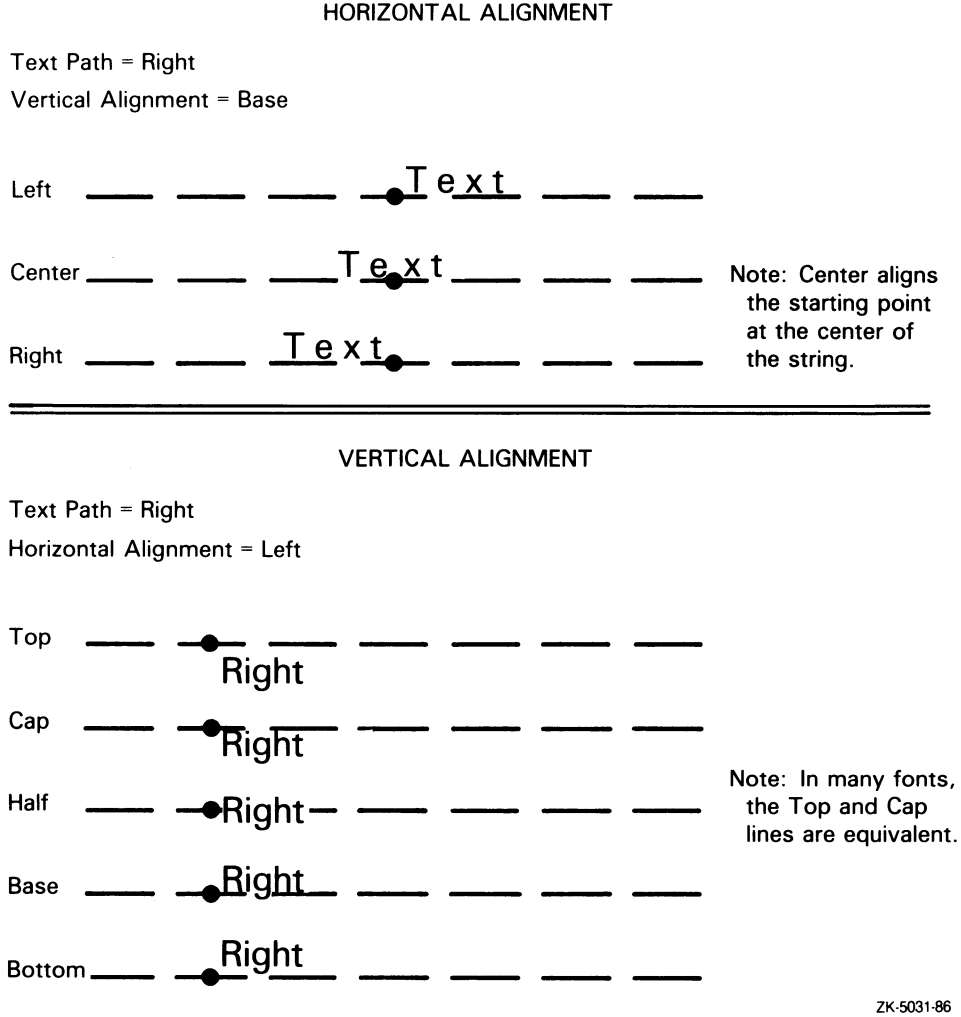
Once you have determined the starting point, the text path (see GKS$SET_TEXT_PATH in this section), and the character up vector (see GKS$SET_TEXT_UPVEC in this section), you have in effect established an imaginary line running through the starting point, on which to output text. At this point, you can use GKS$SET_TEXT_ALIGN to shift the text extent rectangle along this established line.

The two arguments passed to this function establish the horizontal and vertical position of the text extent rectangle on the imaginary text line. For example, you can position the text extent rectangle horizontally so the starting point is to the left, in the center, or to the right of the text extent rectangle.

Not only can you position the text extent rectangle horizontally along the imaginary text line, you can position the rectangle vertically along the same line. For example, you can position the text extent rectangle so that the starting point is aligned with the *top* of the characters in the string, with the *cap* of the characters, with the *half line* of the characters, with the *base line* of the characters, or with the *bottom line* of the characters.

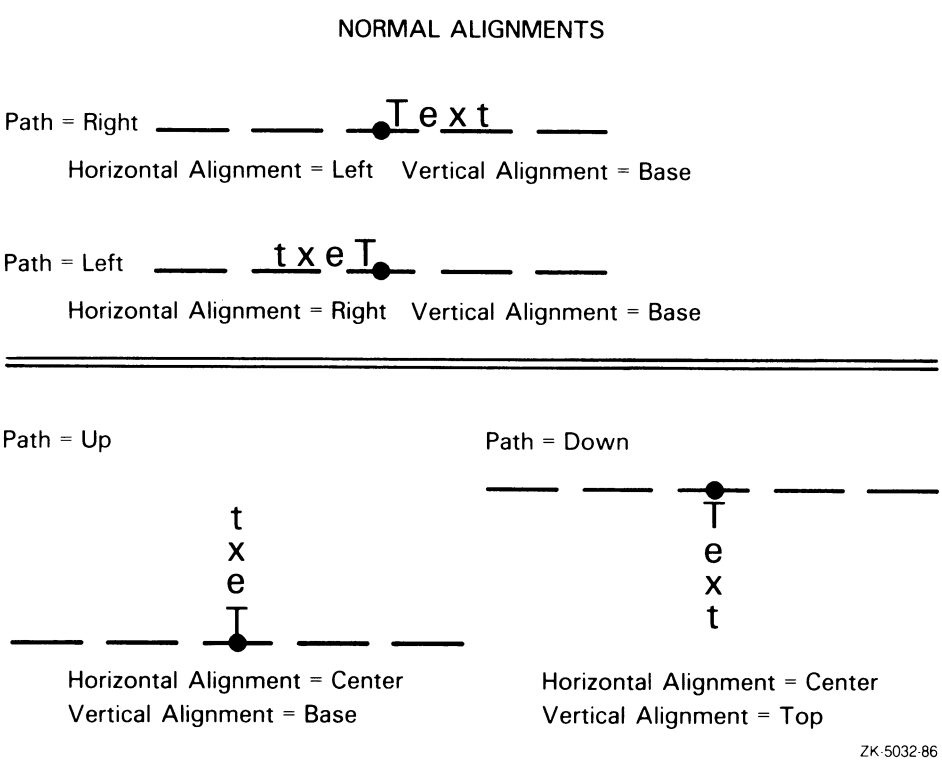
Figure 6–12 illustrates how you can align the text extent rectangle horizontally and vertically. The text path is from right to left and the imaginary text line is illustrated as a dashed line.

Figure 6–12: Horizontal and Vertical Text Alignment



The default horizontal and vertical alignments depend on the text path, and can be explicitly passed using the arguments GKS$K_TEXT_HALIGN_NORMAL and GKS$K_TEXT_VALIGN_NORMAL. Figure 6–13 shows the default values for each of the four text paths.

Figure 6–13: Default Horizontal and Vertical Text Alignments



For more information concerning the text extent rectangle or character output in general, refer to the *DEC GKS User Manual*.

Syntax

GKS$SET_TEXT_ALIGN *(horizontal, vertical)*

GSTXAL *(halign, valign)*

gsettextalign *(align)*

Arguments

horizontal

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the horizontal alignment for text output. The argument can be any of the following values or constants.

Value	Constant	Description
0	GKS$K_TEXT_HALIGN_NORMAL	Normal
1	GKS$K_TEXT_HALIGN_LEFT	Left
2	GKS$K_TEXT_HALIGN_CENTER	Center
3	GKS$K_TEXT_HALIGN_RIGHT	Right

For more information on the use of these values and constants, refer to Figures 6–12 and 6–13.

vertical

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the vertical alignment for text output. The argument can be any of the following values or constants.

Value	Constant	Description
0	GKS$K_TEXT_VALIGN_NORMAL	Normal
1	GKS$K_TEXT_VALIGN_TOP	Top
2	GKS$K_TEXT_VALIGN_CAP	Cap
3	GKS$K_TEXT_VALIGN_HALF	Half
4	GKS$K_TEXT_VALIGN_BASE	Base
5	GKS$K_TEXT_VALIGN_BOTTOM	Bottom

For more information on the use of these values and constants, refer to Figures 6–12 and 6–13.

Error Messages

Error Number	Completion Status Code	Message
–12	DECGKS$_ERROR_NEG_12	Invalid value specified for horizontal component of text alignment in routine ****
–13	DECGKS$_ERROR_NEG_13	Invalid value specified for vertical component of text alignment in routine ****
–20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****

Program Example

Example 6–12 illustrates the use of the function GKS$SET_TEXT_ALIGN. Following the program example, Figure 6–14 illustrates the program's effect on a VT241 workstation.

Example 6–12: Changing the Text Alignment

```
C     This program writes a string to the workstation using the
C     normal text alignments for each of the text paths.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, ONE_PMARK, RED
      REAL LARGER, START_PT_X, START_PT_Y
      DATA WS_ID / 1 /, ONE_PMARK / 1 /, LARGER / 0.07 /,
     * START_PT_X / 0.5 /, START_PT_Y / 0.5 /, RED / 2 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

(1)   CALL GKS$SET_TEXT_HEIGHT( LARGER )
(2)   CALL GKS$SET_PMARK_COLOR_INDEX( RED )
      CALL GKS$SET_PMARK_TYPE( GKS$K_MARKERTYPE_PLUS )

(3)   CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_RIGHT )
      CALL GKS$SET_TEXT_ALIGN( GKS$K_TEXT_HALIGN_NORMAL,
     * GKS$K_TEXT_VALIGN_NORMAL )
      CALL GKS$TEXT( START_PT_X, START_PT_Y, 'TEXT' )
      CALL GKS$POLYMARKER( ONE_PMARK, START_PT_X, START_PT_Y )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

(4)   CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_LEFT )
      CALL GKS$SET_TEXT_ALIGN( GKS$K_TEXT_HALIGN_NORMAL,
     * GKS$K_TEXT_VALIGN_NORMAL )
      CALL GKS$TEXT( START_PT_X, START_PT_Y, 'TEXT' )
      CALL GKS$POLYMARKER( ONE_PMARK, START_PT_X, START_PT_Y )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

(5)   CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_UP )
      CALL GKS$SET_TEXT_ALIGN( GKS$K_TEXT_HALIGN_NORMAL,
     * GKS$K_TEXT_VALIGN_NORMAL )
      CALL GKS$TEXT( START_PT_X, START_PT_Y, 'TEXT' )
      CALL GKS$POLYMARKER( ONE_PMARK, START_PT_X, START_PT_Y )
```

(continued on next page)

Example 6–12 (Cont.): Changing the Text Alignment

```
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

❻     CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_DOWN )
      CALL GKS$SET_TEXT_ALIGN( GKS$K_TEXT_HALIGN_NORMAL,
     * GKS$K_TEXT_VALIGN_NORMAL )
      CALL GKS$TEXT( START_PT_X, START_PT_Y, 'TEXT' )
      CALL GKS$POLYMARKER( ONE_PMARK, START_PT_X, START_PT_Y )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ This code increases the character height so that the string is easy to see.

❷ This code sets the polymarker color and type. A red plus sign marks the text starting point.

❸ Set the text path to GKS$K_TEXT_PATH_RIGHT. Notice that the normal alignment for this path includes horizontal alignment of the starting point to the left and vertical alignment along the base of the letters.

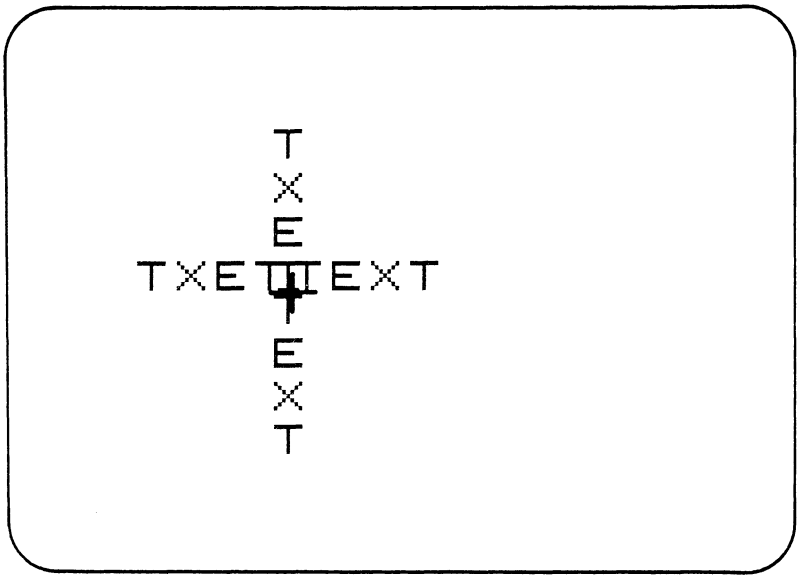
❹ Set the text path to GKS$K_TEXT_PATH_LEFT. Notice that the normal alignment for this path includes horizontal alignment of the starting point to the right and vertical alignment along the base of the letters.

❺ Set the text path to GKS$K_TEXT_PATH_UP. Notice that the normal alignment for this path includes horizontal alignment of the starting point to the center and vertical alignment along the base of the first letter.

❻ Set the text path to GKS$K_TEXT_PATH_DOWN. Notice that the normal alignment for this path includes horizontal alignment to the center and vertical alignment along the top of the first letter.

Figure 6–14 shows the screen of a VT241 terminal after the program has run to completion. The text is in green and the plus sign is red.

Figure 6–14: Changing the Text Alignment—VT241



SET TEXT COLOR INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_TEXT_COLOR_INDEX sets the *current text color index* entry in the DEC GKS state list to the specified value.

Syntax

GKS$SET_TEXT_COLOR_INDEX *(color_index)*

GSTXCI *(cindex)*

gsettextcolourind *(index)*

Arguments

color_index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the text color index. The default value for the text color index entry is the value 1, which designates the default foreground color. For more information concerning predefined color indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
92	GKS$_ERROR_92	Color index is less than zero in routine ****

Program Example

Example 6–13 illustrates the use of the function GKS$SET_TEXT_COLOR_INDEX. Following the program example, Figure 6–15 illustrates the program's effect on a VT241 workstation.

Example 6–13: Changing the Text Color Index

```
C     This program produces a blue text string.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, BLUE
      REAL LARGER, START_PT_X, START_PT_Y
      DATA WS_ID / 1 /, LARGER / 0.07 /, BLUE / 3 /,
     * START_PT_X / 0.5 /, START_PT_Y / 0.5 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❶     CALL GKS$SET_TEXT_HEIGHT( LARGER )

❷     CALL GKS$SET_TEXT_COLOR_INDEX( BLUE )
      CALL GKS$TEXT( START_PT_X, START_PT_Y, 'TEXT' )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

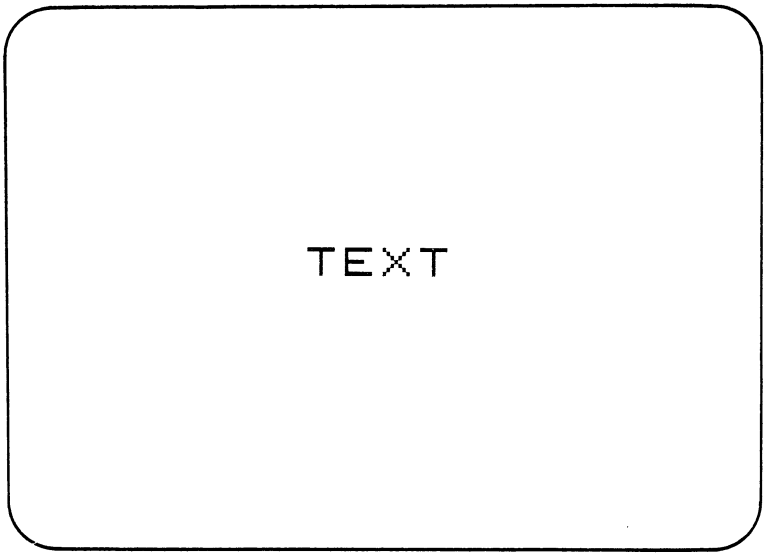
❶ This code increases the character height so that the string is easy to see.

❷ The function, GKS$SET_TEXT_COLOR_INDEX, changes the color from the default color to blue, but only if the text color ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

The VT241 predefines the color index value 3 to be blue. Workstations other than the VT241 may predefine a different representation of color index 3 (a color other than blue).

Figure 6–15 shows the screen of a VT241 terminal after the program has run to completion. The text changed from the default color green to the color blue.

Figure 6–15: Changing the Text Color Index—VT241



ZK-5063-86

SET TEXT EXPANSION FACTOR

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_TEXT_EXPFAC sets the *current character expansion factor* entry in the DEC GKS state list to the specified value. This function alters the width of the generated characters, but not the height.

When DEC GKS calculates the character width using the default character height, the resulting text string is legible. However, certain normalization transformations distort the text. You can either use GKS$SET_TEXT_EXPFAC or GKS$SET_TEXT_HEIGHT to reestablish a legible character width. (For more information concerning transformations, refer to Chapter 7, Transformation Functions.)

Syntax

GKS$SET_TEXT_EXPFAC *(expansion_factor)*

GSCHXP *(efactor)*

gsetcharexpan *(exp)*

Arguments

expansion_factor

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

This argument is the character expansion factor. This value multiplied by the width-to-height ratio specified in the original font specification determines the new character width. The character height remains the same.

The default for the current character expansion factor is the value 1.0, which displays text using the width-to-height ratio specified in the font design.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-28	DECGKS$_ERROR_NEG_28	Invalid value specified for expansion factor in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
77	GKS$_ERROR_77	Character expansion factor is less than or equal to zero in routine ****

Program Example

Example 6–14 illustrates the use of the function GKS$SET_TEXT_EXPFAC. Following the program example, Figure 6–16 illustrates the program's effect on a VT241 workstation.

Example 6–14: Changing the Character Expansion Factor

```
C     This program increases text width by three times the
C     nominal size.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID
      REAL TIMES_THREE, START_PT_X, START_PT_Y, LARGER
      DATA WS_ID / 1 /, TIMES_THREE / 3.0 /,
     * START_PT_X / 0.5 /, START_PT_Y / 0.5 /,
     * LARGER / 0.03 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Make the text easy to read.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( START_PT_X, START_PT_Y, 'TEXT' )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Clear the screen and generate wider characters.
(1)   CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_CONDITIONALLY )
(2)   CALL GKS$SET_TEXT_EXPFAC( TIMES_THREE )
      CALL GKS$TEXT( START_PT_X, START_PT_Y, 'TEXT' )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

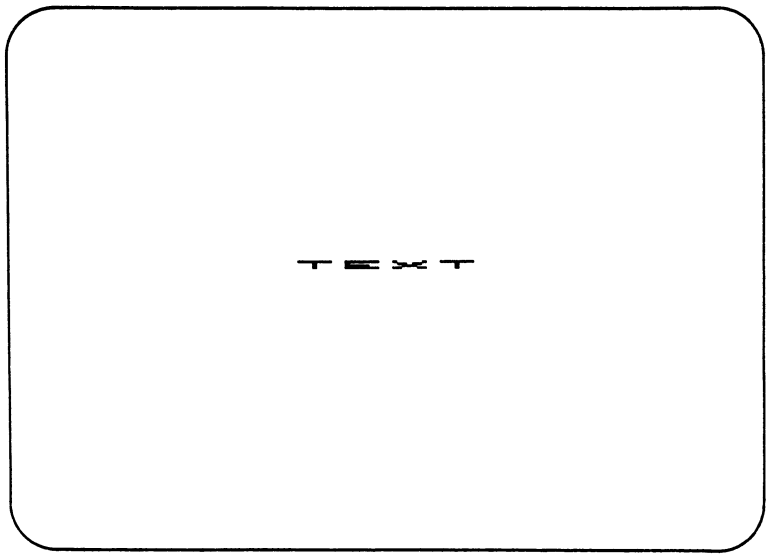
The following numbers correspond to the numbers in the previous example:

❶ Clear the screen conditionally.

❷ The function GKS$SET_TEXT_EXPFAC changes the expansion factor so that text is displayed three times the width-to-height ratio specified in the original font design, but only if the character expansion factor ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Figure 6–16 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–16: Changing the Character Expansion Factor—VT241



ZK-5064-86

SET TEXT FONT AND PRECISION

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_TEXT_FONTPREC sets the *current text font and precision* entry in the DEC GKS state list to the specified value.

When using this function, the types of fonts available for use depend on which precision value you pass as an argument. The values, in order of increasing precision, are as follows:

- String (GKS$K_TEXT_PRECISION_STRING)
- Character (GKS$K_TEXT_PRECISION_CHAR)
- Stroke (GKS$K_TEXT_PRECISION_STROKE)

As the precision increases, the precision of clipping, character size, character spacing, character expansion factor, and the character up vector all improve.

If you specify string precision, and if you specify a starting position for the string that is located outside of the current normalization viewport, the call to GKS$SET_TEXT_FONTPREC causes the entire text string to be clipped. If the starting point for the string is located inside of the current normalization viewport, this function may cause the string to be clipped by character or by stroke depending on the capabilities of the workstation. If you require string precison, you cannot use the DEC GKS software fonts; you can only specify the numbers of the device-dependent fonts available on your particular workstation. For more information concerning the device-dependent fonts available on a workstation, refer to the *DEC GKS Device Specifics Reference Manual.*

If you specify character precision, the call to GKS$SET_TEXT_FONTPREC causes the text string to be clipped at the current normalization viewport on a character-by-character basis. If you require character precison, you cannot use the DEC GKS software fonts; you can only specify the numbers of the device-dependent fonts available on your particular workstation. For more information concerning the fonts available on a workstation, refer to the *DEC GKS Device Specifics Reference Manual.*

If you specify stroke precision, the call to GKS$SET_TEXT_FONTPREC causes the text string to be clipped exactly at the current normalization viewport. This is the highest precision. When using this precision, you may make use of the device-independent fonts that are available on all workstations. For a description of each of these available software fonts and their values, refer to Appendix G, DEC GKS Device-Independent Fonts.

Be aware that all images are clipped at the current workstation window. For more information concerning clipping, refer to Chapter 7, Transformation Functions.

Together, text font and precision specify the display quality of text and the speed at which the text is displayed. Typically, use of a software font in stroke precision produces higher-quality character symbols than use of a hardware font in either character or string precision. However, character and string precision use the workstation character generator, if available, to display text and thus produce the images somewhat faster than stroke precision. Also, since character and string precision are less precise in the application of the other text attributes (for example, height and width), they require less calculation to represent each character in a text string.

The default value for the current text font and precision entry specifies the hardware font value, 1, and string precision.

Syntax

GKS$SET_TEXT_FONTPREC *(font_value, precision_value)*

GSTXFP *(font, precision)*

gsettextfontprec *(txfp)*

Arguments

font_value

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the font value. If you are using the character or string precisions, refer to the *DEC GKS Device Specifics Reference Manual* for more

information. If using stroke precision, refer to Appendix G, DEC GKS Device-Independent Fonts, in this manual.

precision_value

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the precision value. See the function description for detailed information concerning these values. The argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_TEXT_PRECISION_STRING	Lowest precision
1	GKS$K_TEXT_PRECISION_CHAR	Moderate precision
2	GKS$K_TEXT_PRECISION_STROKE	Highest precision

Error Messages

Error Number	Completion Status Code	Message
-14	DECGKS$_ERROR_NEG_14	Invalid value specified for text precision in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-32	DECGKS$_ERROR_NEG_32	Font file for stroke precision text not found or unusable in routine ****
-34	DECGKS$_ERROR_NEG_34	String length less than or equal to 0 in routine ****

Error Number	Completion Status Code	Message
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
75	GKS$_ERROR_75	Text font is equal to zero in routine ****

Program Example

Example 6–15 illustrates the use of the function GKS$SET_TEXT_FONTPREC. Following the program example, Figure 6–17 illustrates the program's effect on a VT241 workstation.

Example 6–15: Changing the Text Font and Precision

```
C     This program changes the default font and precision to
C     stroke/Old English.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, OLD_ENGLISH
      REAL START_PT_X, START_PT_Y, LARGER
      DATA WS_ID / 1 /, START_PT_X / 0.01 /, START_PT_Y / 0.5 /,
     * LARGER / 0.05 /, OLD_ENGLISH / -18 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❶     CALL GKS$SET_TEXT_HEIGHT( LARGER )

❷     CALL GKS$SET_TEXT_FONTPREC( OLD_ENGLISH,
     * GKS$K_TEXT_PRECISION_STROKE )
      CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'THE MORAL KIOSK' )
```

(continued on next page)

Example 6–15 (Cont.): Changing the Text Font and Precision

```
CALL GKS$DEACTIVATE_WS( WS_ID )
CALL GKS$CLOSE_WS( WS_ID )
CALL GKS$CLOSE_GKS()
END
```

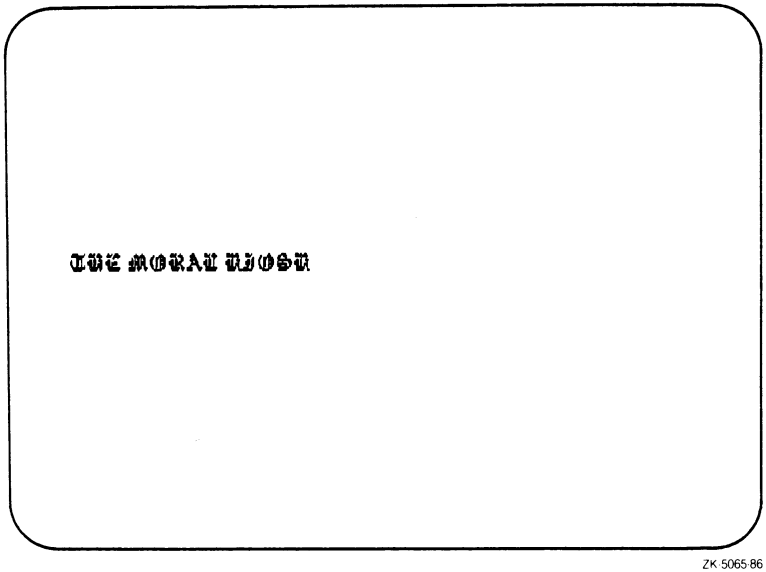
The following numbers correspond to the numbers in the previous example:

❶ This code increases the character height so that the string is easy to see.

❷ The function, GKS$SET_TEXT_FONTPREC, changes the font from the hardware font 1, to the Old English software font -18. GKS$SET_TEXT_FONTPREC also changes the precision from string to stroke. These changes can only occur if the text font and precision ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Figure 6–17 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–17: Changing the Text Font and Precision



ZK-5065-86

SET TEXT HEIGHT

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function, GKS$SET_TEXT_HEIGHT, sets the geometric attribute, *current character height* entry in the DEC GKS state list to the specified world coordinate unit value. DEC GKS uses this value for all subsequent calls to GKS$TEXT until you specify another value.

If you specify a new height to GKS$SET_TEXT_HEIGHT, DEC GKS expands text output to the closest height the workstation is capable of producing. Exercise caution if you change the size of the current normalization window since you may also have to readjust the character height.

Also remember that changing the text height automatically changes the character expansion factor and the character spacing, in proportion to the text height adjustment. (For more information concerning the world coordinate system and transformations, refer to Chapter 7, Transformation Functions.)

Syntax

GKS$SET_TEXT_HEIGHT *(height)*

GSCHH *(height)*

gsetcharheight *(height)*

Arguments

height

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

This argument is the character height; it specifies the character height in world coordinate units. Text height is an absolute value.

The default for the current text height is the world coordinate unit value 0.01. The absolute world coordinate value 0.01 is one percent of the default normalization window height (1.0).

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
77	GKS$_ERROR_77	Character height is less than or equal to zero in routine ****

Program Example

Example 6–16 illustrates the use of the function GKS$SET_TEXT_HEIGHT. Following the program example, Figure 6–18 illustrates the program's effect on a VT241 workstation.

Example 6–16: Changing the Text Height

```
C     This program increases character height.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID
      REAL THIRD_WC_POINT, START_PT_X, START_PT_Y
      DATA WS_ID / 1 /, THIRD_WC_POINT / 0.03 /,
     * START_PT_X / 0.1 /, START_PT_Y / 0.5 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )

❶     CALL GKS$TEXT( START_PT_X, START_PT_Y, 'Life During Wartime' )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

❷     CALL GKS$SET_TEXT_HEIGHT( THIRD_WC_POINT )
❸     CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_ALWAYS )
      CALL GKS$TEXT( START_PT_X, START_PT_Y, 'Life During Wartime' )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

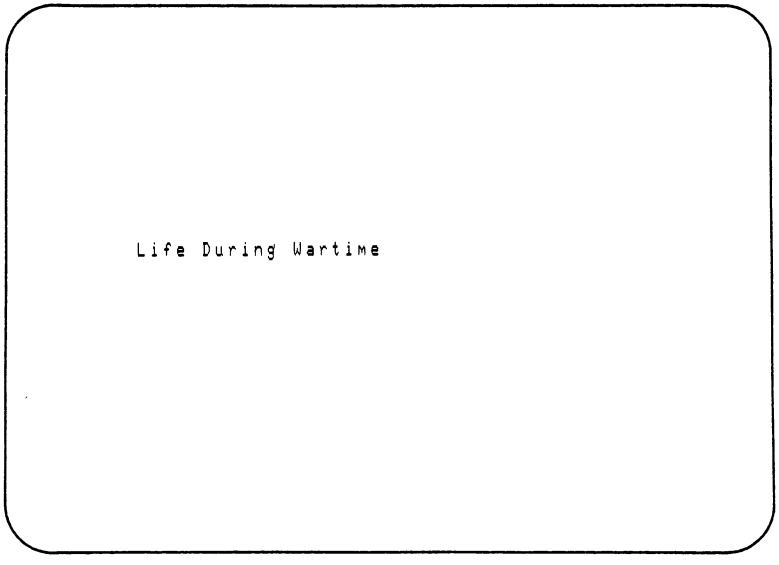
❶ This code outputs text in the default height of 0.01 world coordinate units.

❷ The function GKS$SET_TEXT_HEIGHT changes the value to 0.03 world coordinate units.

❸ This code clears the screen unconditionally. After the next call to GKS$TEXT, text is output to the screen at the new height.

Figure 6–18 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–18: Changing the Text Height—VT241



ZK-5842-HC

SET TEXT INDEX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_TEXT_INDEX establishes the index value pointing into the text bundle table. The text bundle table contains entries for the text font and precision, character expansion factor, character spacing, and text color index attribute values. When calling GKS$TEXT, DEC GKS uses the bundle table only if the corresponding attribute source flag has been set to GKS$K_ASF_BUNDLED.

For a list of the available text bundles for each workstation, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_TEXT_INDEX *(index)*

GSTXI *(tindex)*

gsettextind *(index)*

Arguments

index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the text bundle index. The default bundle index is the value 1. For more information concerning predefined text bundle table indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
72	GKS$_ERROR_72	Text index is invalid in routine ****

Program Example

Example 6–17 illustrates the use of the function GKS$SET_TEXT_INDEX. Following the program example, Figure 6–19 illustrates the program's effect on a VT241 workstation.

Example 6–17: Changing the Text Index

```
C     This program sets the Attribute Source Flags (ASFs) to bundled,
C     and then displays the effects of using the six index values
C     in calls to GKS$SET_TEXT_INDEX.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, FLAGS( 13 ), INCR
      REAL START_PT_X, START_PT_Y, LARGER
      DATA WS_ID / 1 /, START_PT_X / 0.1 /, START_PT_Y / 0.5 /,
     * LARGER / 0.03 /
      CHARACTER*2 STR
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240)
      CALL GKS$ACTIVATE_WS( WS_ID )
```

(continued on next page)

Example 6–17 (Cont.): Changing the Text Index

```
❶       FLAGS( 7 ) = GKS$K_ASF_BUNDLED
        FLAGS( 8 ) = GKS$K_ASF_BUNDLED
        FLAGS( 9 ) = GKS$K_ASF_BUNDLED
        FLAGS( 10 ) = GKS$K_ASF_BUNDLED
        CALL GKS$SET_ASF( FLAGS )

        CALL GKS$SET_TEXT_HEIGHT( LARGER )

❷       DO 200 INCR = 1, 6, 1
        CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_ALWAYS )
        CALL GKS$SET_TEXT_INDEX( INCR )
        CALL GKS$TEXT( START_PT_X, START_PT_Y,
       * 'Family of Max Desir' )
❸       CALL LIB$CVT_DX_DX( %DESCR( INCR ), %DESCR( STR ) )
        CALL GKS$TEXT( .1, .3, 'Index:  ')
        CALL GKS$TEXT( .4, .3, %DESCR( STR ) )
C       Release deferred output. Pause.  Type RETURN when you are finished
C       viewing the picture.
        CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
        READ(5,*)

200     CONTINUE

        CALL GKS$DEACTIVATE_WS( WS_ID )
        CALL GKS$CLOSE_WS( WS_ID )
        CALL GKS$CLOSE_GKS()
        END
```

The following numbers correspond to the numbers in the previous example.

❶ This code initializes the elements of the array that affect all of the nongeometric polymarker attributes. This code sets each ASF to GKS$K_ASF_BUNDLED. FLAGS(7) corresponds to the text font and precision; FLAGS(8) corresponds to the character expansion factor; FLAGS(9) corresponds to the character spacing; and, FLAGS(10) corresponds to the text color index.

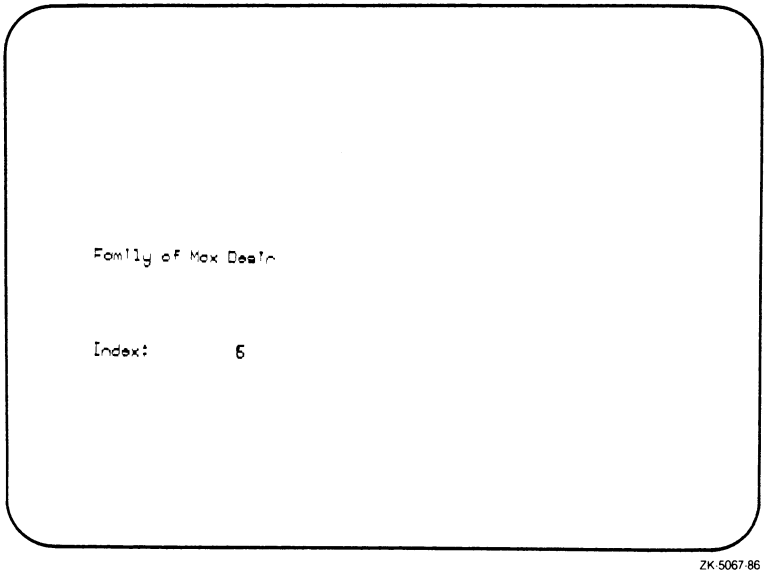
See the GKS$SET_ASF function description in this chapter for more information.

❷ This code displays the text using the six text index values available on the VT241. This code writes the index value that produced the text, in the lower left portion of the screen.

❸ This VMS Run-Time Library Routine translates the variable INCR to a text string so that GKS$TEXT can write the text index value to the screen.

Figure 6–19 shows the screen of a VT241 terminal after the program has run to completion. The color of the text is blue.

Figure 6–19: Changing the Text Index—VT241



SET TEXT PATH

Operating States: GKOP, WSOP, WSAC, SGOP

Description

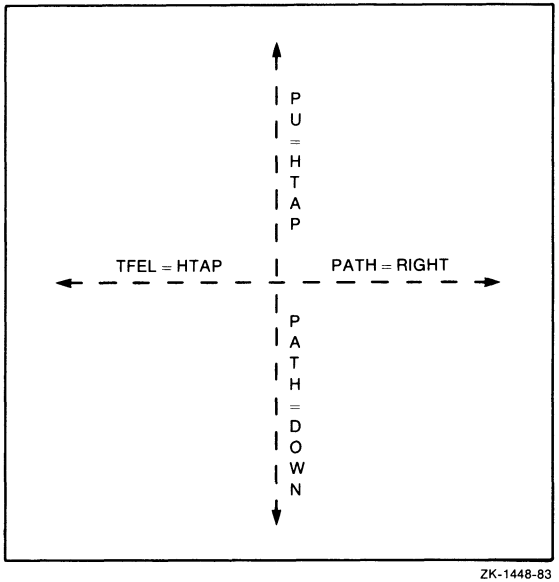
The function GKS$SET_TEXT_PATH sets the geometric attribute *current text path* entry in the DEC GKS state list to be the writing direction for the display of text. DEC GKS uses this value for all subsequent calls to GKS$TEXT until you specify another value.

Once you have determined the starting point and the character up vector (see GKS$SET_TEXT_UPVEC in this section), you have in effect established an imaginary line running through the starting point to use when generating text primitives. You can output your text string with your aligned letter starting at the starting point (see GKS$SET_TEXT_ALIGN in this section). According to the current text path, the string either reads to the right along the imaginary line (the default), to the left along the imaginary line, upwards in a perpendicular direction from the imaginary text line, or downwards in a perpendicular direction from the imaginary line.

If using the default text alignment (see GKS$SET_TEXT_ALIGN), DEC GKS places the first letter of this string at the starting point, and subsequent letters are written along the imaginary text line in the direction specified by a call to this function. The default text path is left to right along the imaginary text line (GKS$K_TEXT_PATH_RIGHT).

Figure 6–20 illustrates the writing direction for each value of text path. The figure assumes (0.0, 1.0) as the character up vector.

Figure 6–20: Text Path Directions



Syntax

GKS$SET_TEXT_PATH *(text_path)*

GSTXP *(text_path)*

gsettextpath *(text_path)*

Arguments

text_path

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the text path. The argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_TEXT_PATH_RIGHT	Left to right
1	GKS$K_TEXT_PATH_LEFT	Right to left
2	GKS$K_TEXT_PATH_UP	Bottom to top
3	GKS$K_TEXT_PATH_DOWN	Top to bottom

Error Messages

Error Number	Completion Status Code	Message
-15	DECGKS$_ERROR_NEG_15	Invalid value specified for text path in routine ***
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****

Program Example

Example 6–18 illustrates the use of the function GKS$SET_TEXT_PATH. Following the program example, Figure 6–21 illustrates the program's effect on a VT241 workstation.

Example 6–18: Changing the Text Path

```
C     This program shows each of the four text paths.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID
      REAL LARGER, START_PT_X, START_PT_Y
      DATA WS_ID / 1 /, LARGER / 0.05 /,
     * START_PT_X / 0.5 /, START_PT_Y / 0.5 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❶     CALL GKS$SET_TEXT_HEIGHT( LARGER )

❷     CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_LEFT )
❸     CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'Burning' )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

      CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_DOWN )
      CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'Down' )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

      CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_RIGHT )
      CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'The' )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

      CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_UP )
      CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'House' )
```

(continued on next page)

Example 6–18 (Cont.): Changing the Text Path

```
CALL GKS$DEACTIVATE_WS( WS_ID )
CALL GKS$CLOSE_WS( WS_ID )
CALL GKS$CLOSE_GKS()
END
```

The following numbers correspond to the numbers in the previous example:

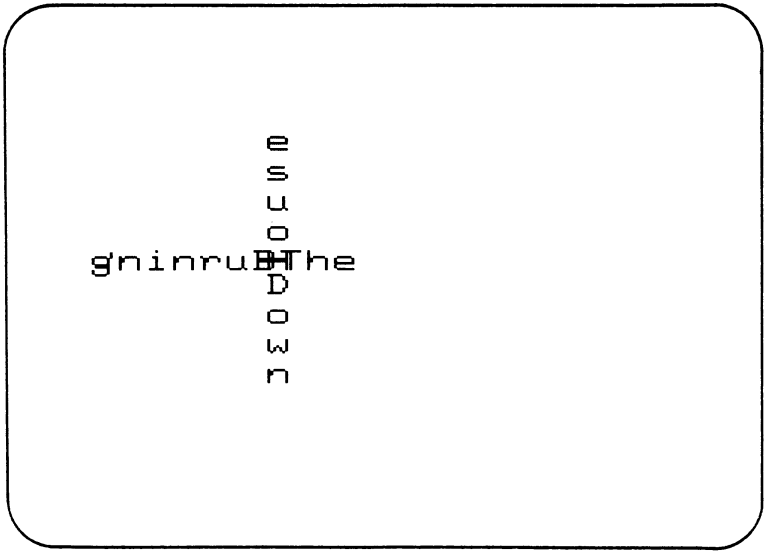
❶ This code increases the character height so that the string is easy to see.

❷ Set the text path (in this case, to the left).

❸ Output the text string.

Figure 6–21 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–21: Changing the Text Path—VT241



ZK-5068-86

SET TEXT SPACING

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_TEXT_SPACING sets the *current text spacing* entry in the DEC GKS state list to the specified value.

DEC GKS measures the spacing between characters as a fraction of the character height; adjusting character height automatically proportionately adjusts spacing. The character spacing value 0.0 places the character bodies next to each other without any separating space contained in the font specification for the letter bodies. Whether or not the characters actually touch depends on the type of font you are using. Positive spacing values increase the space between characters; negative values decrease the space. Using negative spacing values, it is possible to overlap characters, or to actually reverse the text so that characters are written in the opposite direction.

Syntax

GKS$SET_TEXT_SPACING *(spacing_percentage)*

GSCHSP *(spacing)*

gsetcharspace *(spacing)*

Arguments

spacing_percentage

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

This argument is the character spacing factor. This value is a percentage of the current character height. The default for the current character spacing entry is the value 0.0, which displays text with adjacent character bodies.

Error Messages

Error Number	Completion Status Code	Message
–20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****

Program Example

Example 6–19 illustrates the use of the function GKS$SET_TEXT_SPACING. Following the program example, Figure 6–22 illustrates the program's effect on a VT241 workstation.

Example 6–19: Changing the Character Spacing

```
C     This program decreases the character spacing enough so that
C     the characters overlap slightly.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID
      REAL START_PT_X, START_PT_Y, LARGER, OVERLAP
      DATA WS_ID / 1 /, START_PT_X / 0.2 /, START_PT_Y / 0.5 /,
     * LARGER / 0.05 /, OVERLAP / -0.3 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❶     CALL GKS$SET_TEXT_HEIGHT( LARGER )

❷     CALL GKS$SET_TEXT_SPACING( OVERLAP )
      CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'Relentless Cookout' )
```

(continued on next page)

Example 6–19 (Cont.): Changing the Character Spacing

```
CALL GKS$DEACTIVATE_WS( WS_ID )
CALL GKS$CLOSE_WS( WS_ID )
CALL GKS$CLOSE_GKS()
END
```

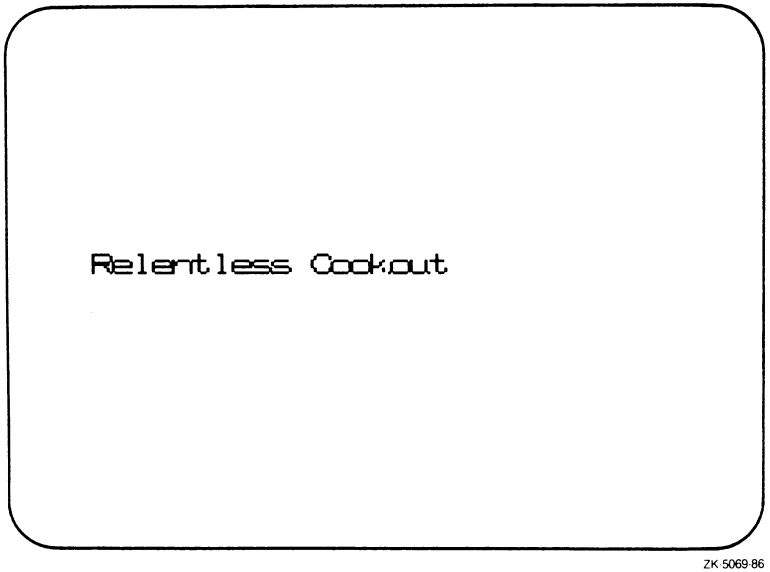
The following numbers correspond to the numbers in the previous example:

❶ This code increases the character height so that the string is easy to see.

❷ The function GKS$SET_TEXT_SPACING changes the spacing from the default spacing to overlapping characters (a negative character spacing). These changes can occur only if the character spacing ASF is set to GKS$K_ASF_INDIVIDUAL (the default setting).

Figure 6–22 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–22: Changing the Character Spacing—VT241



ZK-5069-86

SET TEXT UP VECTOR

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_TEXT_UPVEC sets the geometric attribute *current character up vector* entry in the DEC GKS state list to the specified value. DEC GKS uses this value for all subsequent calls to GKS$TEXT until you specify another value.

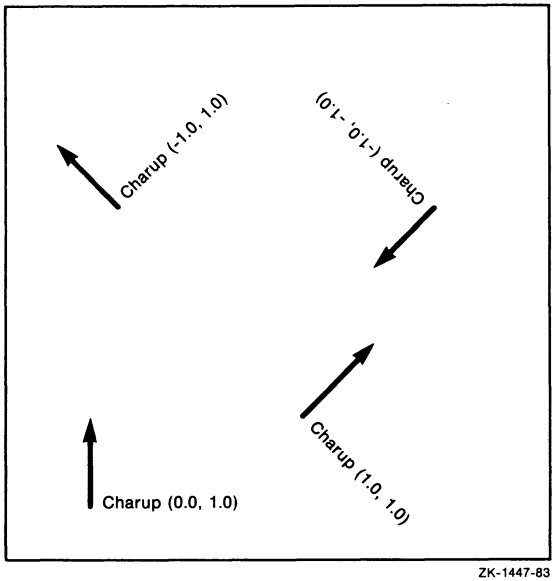
When you call GKS$TEXT, you specify the starting point for the text. In order to establish an imaginary line on which to output text, you must establish an upward direction. Once an upward direction has been established, DEC GKS draws an imaginary line that is perpendicular to this upward direction that runs through the starting point. This perpendicular line is the imaginary line on which you can output text, by the positioning of the text extent rectangle.

You specify the upward direction for character placement as a directional vector. The vector begins at the starting point and proceeds in the direction of the current character up vector entry. You establish the character up vector by specifying a slope for the line.

For example, if you specify the world coordinates unit values (1.0, 1.0) as the character up vector, the up direction for the display of text follows the line passing from the starting point to the point one point above and one point to the right of the starting point. This would correspond to a 45-degree angle of rotation. Specifying the values (200.0, 200.0), or the values (5.0, 5.0), is equivalent to specifying (1.0, 1.0).

Figure 6–23 illustrates some of the possible values of the character up vector.

Figure 6–23: Examples of Character Up Vector Entries



The initial value for the current character up vector entry is (0.0, 1.0), which orients text perpendicular to the X-axis and parallel to the Y-axis, if the current character path is GKS$K_TEXT_PATH_RIGHT or GKS$K_TEXT_PATH_LEFT.

Syntax

GKS$SET_TEXT_UPVEC *(x_vector_entry, y_vector_entry)*

GSCHUP *(x_vector, y_vector)*

gsetcharup *(charup)*

Arguments

x_vector_entry
y_vector_entry

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments are the X and Y unit values that establish the character up vector entry. Specifically, these values specify the slope of the character up vector.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****
79	GKS$_ERROR_79	Length of character up vector is zero in routine ****

Program Example

Example 6–20 illustrates the use of the function GKS$SET_TEXT_UPVEC. Following the program example, Figure 6–24 illustrates the program's effect on a VT241 workstation.

Example 6–20: Changing the Up Character Vector

```
C     This program shifts the default character up vector
C     to the left.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID
      REAL LARGER, START_PT_X, START_PT_Y, VECTOR_X, VECTOR_Y
      DATA WS_ID / 1 /, LARGER / 0.05 /, START_PT_X / 0.5 /,
     * START_PT_Y / 0.5 /, VECTOR_X / -1.0 /, VECTOR_Y / 1.0 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❶     CALL GKS$SET_TEXT_UPVEC( VECTOR_X, VECTOR_Y )

❷     CALL GKS$SET_TEXT_HEIGHT( LARGER )

❸     CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_LEFT )
❹     CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'John' )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

      CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_DOWN )
      CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'Paul' )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

      CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_RIGHT )
      CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'George' )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
```

(continued on next page)

Example 6–20 (Cont.): Changing the Up Character Vector

```
      CALL GKS$SET_TEXT_PATH( GKS$K_TEXT_PATH_UP )
      CALL GKS$TEXT( START_PT_X, START_PT_Y,
     * 'Ringo' )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ This code alters the character up vector so that all text tilts to the left.

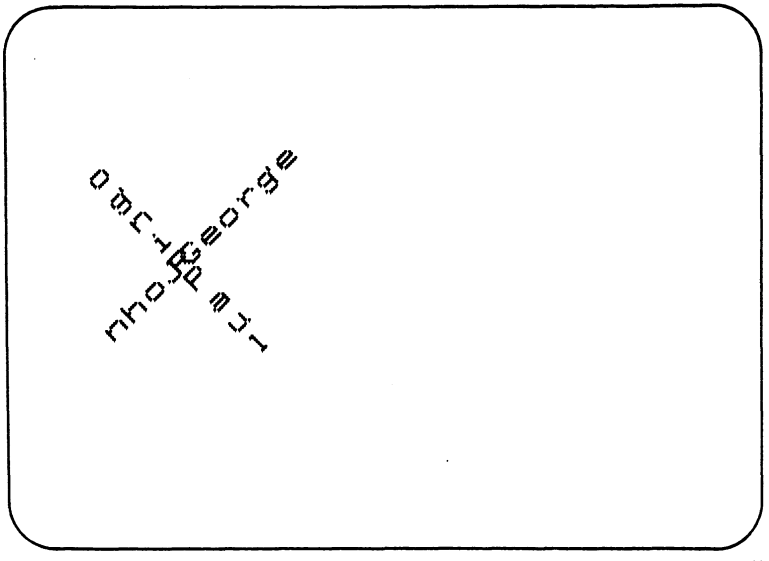
❷ This code increases the character height so that the string is easy to see.

❸ Set the text path (in this case, to the left).

❹ Generate the text string.

Figure 6–24 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–24: Changing the Up Character Vector—VT241



ZK-5070-86

Aspect Source Flag Function

This section describes the aspect source flags (ASFs), which are nongeometric attributes.

SET ASPECT SOURCE FLAGS

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_ASF specifies to DEC GKS whether to use the bundled or the individual method for designating each of the nongeometric output attributes.

There are thirteen nongeometric attribute source flags (ASF), numbered 1 to 13. You pass an array with thirteen elements to GKS$SET_ASF. If the value in the corresponding element is GKS$K_ASF_INDIVIDUAL, DEC GKS uses the individual attribute setting. If the value in the corresponding element is GKS$K_ASF_BUNDLED, DEC GKS uses the bundle table index to find the attribute setting.

The initial value for each ASF is individual, which causes the output functions to use the current individual value for each nongeometric attribute. Remember that when specified individually, attributes are workstation independent; when specified as a bundle, the attributes are workstation dependent. For instance, most workstations provide a fill area bundle index 1, but the resulting fill area can look different on each workstation. For more information concerning the bundle table indexes available for your workstation, refer to the *DEC GKS Device Specifics Reference Manual*.

Syntax

GKS$SET_ASF *(flags)*

GSASF *(flags)*

gsetasf *(asfs)*

Arguments

flags

data type: **array (integer)**
access: **read-only**
mechanism: **by reference**

This argument is the array of the thirteen attribute source flags (ASFs). There exists one element for each of the nongeometric output attributes, as follows:

Number	Nongeometric Attribute
1	line type
2	line width scale factor
3	polyline color index
4	marker type
5	marker size scale factor
6	polymarker color index
7	text font and precision
8	character expansion factor
9	character spacing
10	text color index
11	fill area interior style
12	fill area style index
13	fill area color index

Error Messages

Error Number	Completion Status Code	Message
-10	DECGKS$_ERROR_NEG_10	Invalid value specified for ASF in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS must be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****

Program Example

Refer to Example 6–5 in this chapter for a program example using a call to GKS$SET_ASF.

Representation Functions

The DEC GKS functions described in this section define or change the nongeometric attributes associated with a given bundle table index. These attributes comprise the index's *representation*. Bundle representations are unique to each workstation (device dependent).

Notice that DEC GKS must be in the GKS$K_WSOP state in order for you to call these functions. For more information concerning operating states, refer to Chapter 4, Control Functions.

A list of the different nongeometric representation types follows:

- Color representation
- Fill representation
- Pattern representation
- Polyline representation
- Polymarker representation
- Text representation

SET COLOR REPRESENTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_COLOR_REP allows you to redefine an existing color index representation, or to define a new representation, by specifying the red, green, and blue color intensities associated with a specified bundle index. The workstation maps the color you specify to the nearest available color the workstation can produce.

All workstations define default color table entry indexes 0 and 1. By default, the value 0 corresponds to the default background color (the color of an empty display surface), and the value 1 corresponds to the default foreground color. Also by default, the values greater than the value 1 correspond to alternative foreground colors.

Depending on the capabilities of your workstation, a call to this function may cause DEC GKS to implicitly regenerate the workstation surface. For information concerning implicit regeneration, refer to Chapter 4, Control Functions.

Attribute values passed to this function must be valid for the specified workstation. For information, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_COLOR_REP *(workstation_id, color_index, red_intensity, green_intensity, blue_intensity)*

GSCR *(workstation_id, cindex, red_i, green_i, blue_i)*

gsetcolourrep *(workstation_id, index, rep)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in Chapter 4, Control Functions).

color_index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the color index value. By specifying a value to this function, you are redefining the associated color by specifying a blend of red, green, and blue intensities previously associated with this color index.

red_intensity
green_intensity
blue_intensity

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

These arguments are the red, green, and blue intensities that form the desired color. RGB values must fall within the range 0.0 to 1.0, or DEC GKS generates an error. For more information concerning these intensities, refer to the *DEC GKS Device Specifics Reference Manual*.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
93	GKS$_ERROR_93	Color index is invalid in routine ****
96	GKS$_ERROR_96	Color is outside range [0,1] in routine ****

Program Example

Example 6–21 illustrates the use of the function GKS$SET_COLOR_REP. Following the program example, Figure 6–25 illustrates the program's effect on a VT241 workstation.

Example 6–21: Changing the Color Representation

```
C     This program changes the fill color of a triangle from
C     the color blue to the color pink.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, BLUE, NUM_POINTS
      REAL PX( 3 ), PY( 3 ), RED_INTENS, GREEN_INTENS, BLUE_INTENS
      DATA WS_ID / 1 /, BLUE / 3 /, NUM_POINTS / 3 /,
     * RED_INTENS / 0.6258 /, GREEN_INTENS / 0.2142 /,
     * BLUE_INTENS / 0.2142 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
❶     DATA PX  /.1, .9, .1/
      DATA PY  /.1, .9, .9/

❷     CALL GKS$SET_FILL_COLOR_INDEX( BLUE )
      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )
      CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

❸     CALL GKS$SET_COLOR_REP( WS_ID, BLUE,
     * RED_INTENS, GREEN_INTENS, BLUE_INTENS )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .1).

❷ The function, GKS$SET_FILL_COLOR_INDEX, changes the color from the default color to blue. By default, the fill area color ASF is set to GKS$K_ASF_INDIVIDUAL.

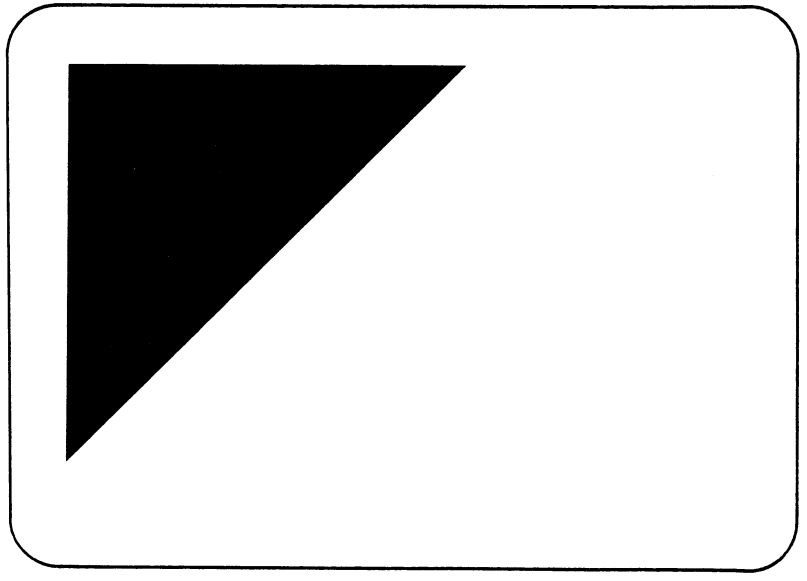
Workstations other than the VT241 may predefine a different representation of color index 3 (a color other than blue).

❸ As soon as you call GKS$SET_COLOR_REP, DEC GKS dynamically changes the blue triangle to a pink triangle.

Other workstations may not be able to dynamically alter the workstation surface, and may need to implicitly regenerate the surface to make a change to a color table entry. For information concerning implicit regenerations, refer to Chapter 4, Control Functions.

Figure 6–25 shows the screen of the VT241 terminal after the program has run to completion.

Figure 6–25: Changing the Color Representation—VT241



SET FILL AREA REPRESENTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_FILL_REP allows you to redefine an existing fill area bundle table index representation, or to define a new fill area bundle table index value, by specifying the fill area interior style, fill area style index value, and fill area color index associated with the specified bundle index.

Depending on the capabilities of your workstation, a call to this function may cause DEC GKS to implicitly regenerate the workstation surface. For information concerning implicit regeneration, refer to Chapter 4, Control Functions.

Attribute values passed to this function must be valid for the specified workstation. For information, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_FILL_REP *(workstation_id, fill_index, interior_style, style_index, color_index)*

GSFAR *(workstation_id, index, style, sindex, cindex)*

gsetfillrep *(workstation_id, index, rep)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in Chapter 4, Control Functions).

fill_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the fill area bundle table index value. By specifying a value to this function, you are redefining the interior style, style index, and color index entries in the associated bundle table. See GKS$SET_FILL_INDEX in this section for more information.

interior_style

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the interior style index value to be associated with the specified fill area index value. The argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_INTSTYLE_HOLLOW	Use an outline.
1	GKS$K_INTSTYLE_SOLID	Use color.
2	GKS$K_INTSTYLE_PATTERN	Use a pattern.
3	GKS$K_INTSTYLE_HATCH	Use crossed or parallel lines.

See GKS$SET_FILL_INT_STYLE in this section for more information.

style_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the fill area style index. If you specified hollow or solid for the interior style argument, DEC GKS ignores this argument. For more information concerning the possible fill area style indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

color_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the fill area color index. For more information concerning the possible fill area color indexes, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
80	GKS$_ERROR_80	Fill area index is invalid in routine ****
83	GKS$_ERROR_83	Specified fill area interior style is not supported on this workstation in routine ****

Error Number	Completion Status Code	Message
85	GKS$_ERROR_85	Specified pattern index is invalid in routine ****
86	GKS$_ERROR_86	Specified hatch style is not supported on this workstation in routine ****
93	GKS$_ERROR_93	Color index is invalid in routine ****

Program Example

Example 6–22 illustrates the use of the function GKS$SET_FILL_REP. Following the program example, Figure 6–26 illustrates the program's effect on a VT241 workstation.

Example 6–22: Changing the Fill Area Representation

```
C     This program sets the Attribute Source Flags (ASFs) to bundled,
C     shows the fill area corresponding to the index 2, and then
C     changes the attributes associated with fill area index 2, using
C     GKS$SET_FILL_REP.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, BLUE, NUM_POINTS, FILL_INDEX,
     * VERT_LINES, FLAGS( 13 ), SEG_NAME
      DATA WS_ID / 1 /, NUM_POINTS / 3 /, FILL_INDEX / 2 /,
     * VERT_LINES / -5 /, BLUE / 3 /, SEG_NAME / 1 /
❶     DATA PX  /.1, .9, .1/
      DATA PY  /.1, .9, .9/
      REAL PX( 3 ), PY( 3 )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     FLAGS( 11 ) = GKS$K_ASF_BUNDLED
      FLAGS( 12 ) = GKS$K_ASF_BUNDLED
      FLAGS( 13 ) = GKS$K_ASF_BUNDLED
      CALL GKS$SET_ASF( FLAGS )
```

(continued on next page)

Example 6–22 (Cont.): Changing the Fill Area Representation

```
C     Put all output in a segment.
      CALL GKS$CREATE_SEG( SEG_NAME )
❸     CALL GKS$SET_FILL_INDEX( FILL_INDEX )
❹     CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

      CALL GKS$SET_FILL_REP( WS_ID, FILL_INDEX,
     * GKS$K_INTSTYLE_HATCH, VERT_LINES, BLUE )

C     Cause a regeneration of the screen to see the change on a VT241.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .1).

❷ This code initializes the elements of the array that affect all of the non-geometric fill area attributes. This code sets each ASF to GKS$K_ASF_BUNDLED. FLAGS(11) corresponds to the current fill area interior style; FLAGS(12) corresponds to the current fill area style index; and, FLAGS(13) corresponds to the fill area color index.

See the GKS$SET_ASF function description in this chapter for more information.

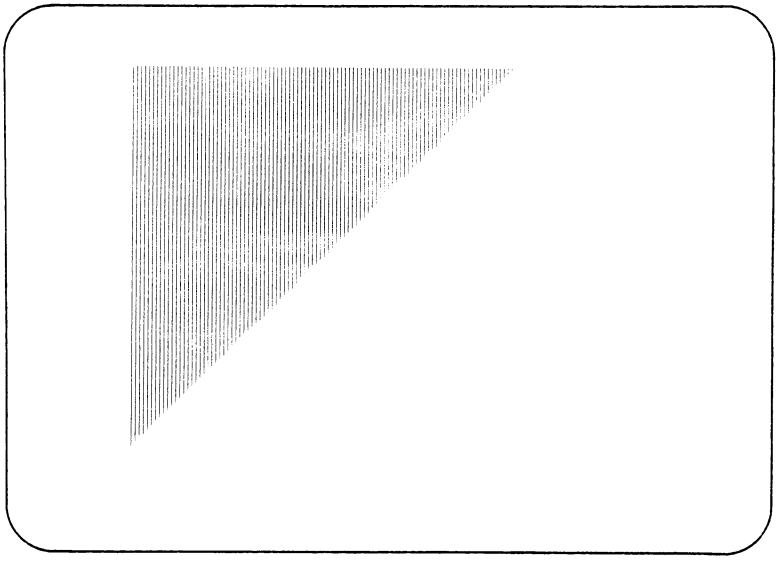
❸ On a VT241, setting the fill area bundle table index to the value 2 specifies a fill area that is solid red.

❹ On a VT241, calling GKS$SET_FILL_REP causes an implicit regeneration that is suppressed by the workstation (by default). The attribute changes are not made and the screen is out of date. You need to call GKS$UPDATE_WS to update the surface of the workstation.

If your workstation requires an implicit regeneration to implement changes to fill area representation but does not suppress the regeneration by default, the workstation redraws only the visible segments on the workstation surface. Output primitives not contained in segments are lost. For a complete discussion of implicit regeneration, refer to Chapter 4, Control Functions.

Figure 6–26 shows the VT241 surface after the program was executed. The color of the triangle changed from red to blue.

Figure 6–26: Changing the Fill Area Representation—VT241



ZK-5845-HC

SET PATTERN REPRESENTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_PAT_REP allows you to redefine an existing pattern bundle table index representation, or to define a new pattern bundle table index value, by specifying the number of cells high, the number of cells wide, and an array containing each cell's color index fill area, associated with the specified bundle index.

Depending on the capabilities of your workstation, a call to this function may cause DEC GKS to implicitly regenerate the workstation surface. For information concerning implicit regeneration, refer to Chapter 4, Control Functions.

Attribute values passed to this function must be valid for the specified workstation. For information, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PAT_REP *(workstation_id, pattern_index, offset_column_number, offset_row_number, num_columns, color_ind_array)*

GSPAR *(workstation_id, pindex, dim_x, dim_y, scol, srow, ncols, nrows, cindex)*

gsetpatrep *(workstation_id, index, rep)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in Chapter 4, Control Functions).

pattern_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the pattern bundle table index value. By specifying a value to this function, you are redefining the height, width, and color previously associated with this pattern bundle table index.

offset_column_number
offset_row_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

These arguments are the offset into the color index array. You can begin mapping color index values from the interior of the array, if you desire.

The offset determines the number of array columns and rows that you specify as arguments to GKS$SET_PAT_REP. For instance, if the offset is the first element of the array, you can specify the full dimensions of the color index array as the "number of columns to map" and the "number of rows to map."

For a detailed discussion of this argument, refer to the GKS$CELL_ARRAY arguments in Chapter 5, Output Functions.

num_columns
num_rows

data type: **integer**
access: **read-only**
mechanism: **by reference**

These arguments specify the number of rows and columns, beginning at the offset element, to map from the color index array to the pattern. For a detailed discussion of this argument, refer to the GKS$CELL_ARRAY argument descriptions in Chapter 5, Output Functions.

color_index_array

data type: **2D array (integer)**
access: **read-only**
mechanism: **by descriptor**

This argument is the array containing the color index values for each individual cell in the pattern. The array must have at least the dimensions that you specified as the height and width arguments.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****

Error Number	Completion Status Code	Message
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
85	GKS$_ERROR_85	Specified pattern index is invalid in routine ****
90	GKS$_ERROR_90	Interior style PATTERN is not supported on this workstation in routine ****
91	GKS$_ERROR_91	Dimensions of color array are invalid in routine ****
93	GKS$_ERROR_93	Color index is invalid in routine ****

Program Example

Example 6–23 illustrates the use of the function GKS$SET_PAT_REP. Following the program example, Figure 6–27 illustrates the program's effect on a VT241 workstation.

Example 6–23: Changing the Pattern Representation

```
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, BLUE,
(1)  * FILL_INDEX, PAT_INDEX, NUM_ROWS, NUM_COLS, PAT_ARRAY ( 2,2 ),
     * SEG_NAME, OFFSET_COL, OFFSET_ROW
      DATA WS_ID / 1 /, NUM_POINTS / 4 /,
     * FILL_INDEX / 8 /, PAT_INDEX / 5 /, BLUE / 3 /,
     * NUM_ROWS / 2 /, NUM_COLS / 2 /,
(2)  * PAT_ARRAY / 3,2, 2,3 /,
     * SEG_NAME / 1 /, OFFSET_COL / 1 /, OFFSET_ROW / 1 /
(3)   DATA PX  /.1, .9, .9, .1 /
      DATA PY  /.1, .1, .9, .9 /
      REAL PX( 4 ), PY( 4 )
      INTEGER FLAGS( 13 )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      FLAGS( 11 ) = GKS$K_ASF_BUNDLED
      FLAGS( 12 ) = GKS$K_ASF_BUNDLED
      FLAGS( 13 ) = GKS$K_ASF_BUNDLED
      CALL GKS$SET_ASF( FLAGS )

C     Store the output in a segment.
      CALL GKS$CREATE_SEG( SEG_NAME )
(4)   CALL GKS$SET_FILL_INDEX( FILL_INDEX )
(5)   CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

(6)   CALL GKS$SET_PAT_REP( WS_ID, PAT_INDEX, OFFSET_COL,
     * OFFSET_ROW, NUM_COLS, NUM_ROWS, %DESCR( PAT_ARRAY ) )
```

(continued on next page)

Example 6–23 (Cont.): Changing the Pattern Representation

```
C     Update the screen to reflect the change.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )
      READ(5,*)

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The fill area bundle table number 8 contains an entry for the pattern index. For the VT241, fill area bundle number 8 specifies an associated pattern index value of 5.

❷ Define the pattern array. The pattern consists of alternating rows of blue and red combinations. Other workstations may define other index values for the colors blue and red. For more information, refer to the *DEC GKS Device Specifics Reference Manual.*

❸ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .1).

❹ This code initializes the elements of the array that affect all of the nongeometric polymarker attributes. This code sets each ASF to GKS$K_ASF_BUNDLED. FLAGS(11) corresponds to the current fill area interior style; FLAGS(12) corresponds to the current fill area style index; and, FLAGS(13) corresponds to the fill area color index.

See the GKS$SET_ASF function description in this chapter for more information.

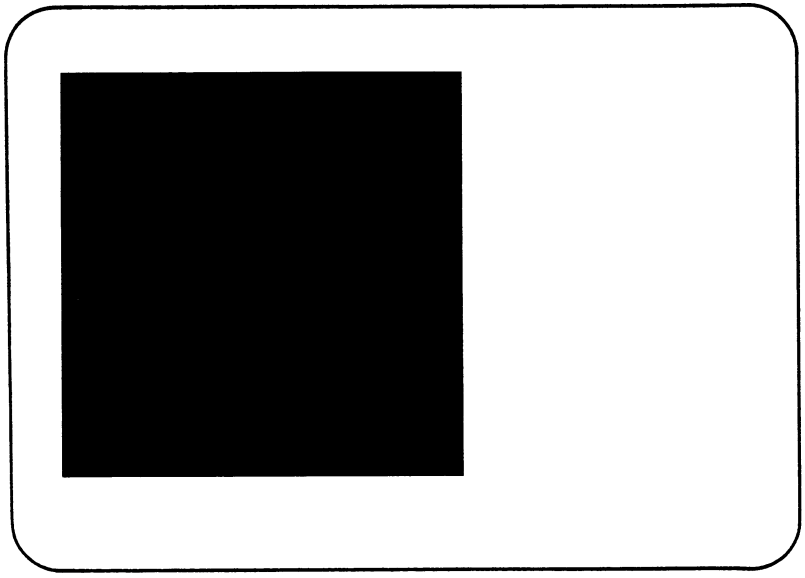
❺ On a VT241, setting the fill area bundle table index to the value 8 specifies a dark red pattern for the fill area style.

❻ On a VT241, calling GKS$SET_PAT_REP causes an implicit regeneration that is suppressed by the workstation (by default). The attribute changes are not made and the screen is out of date. You need to call GKS$UPDATE_WS to update the surface of the workstation.

If your workstation requires an implicit regeneration to implement changes to pattern representation but does not suppress the regeneration by default, the workstation redraws only the visible segments on the workstation surface. Output primitives not contained in segments are lost. For a complete discussion of implicit regeneration, refer to Chapter 4, Control Functions.

Figure 6–27 shows the VT241 surface after the program was executed. The color of the triangle changed from a red and black pattern to a red and blue pattern.

Figure 6–27: Changing the Pattern Representation—VT241



ZK-5076-86

SET POLYLINE REPRESENTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_PLINE_REP allows you to redefine an existing polyline bundle table index representation, or to define a new polyline bundle table index value, by specifying the line type, the line width, and the line color index associated with the specified bundle index.

Depending on the capabilities of your workstation, a call to this function may cause DEC GKS to implicitly regenerate the workstation surface. For information concerning implicit regeneration, refer to Chapter 4, Control Functions.

Attribute values passed to this function must be valid for the specified workstation. For information, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PLINE_REP *(workstation_id, polyline_index, line_type, line_width, color_index)*

GSPLR *(workstation_id, pindex, ltype, lwidth, cindex)*

gsetlinerep *(workstation_id, index, rep)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in Chapter 4, Control Functions).

polyline_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the polyline bundle table index value. By specifying a value to this function, you are redefining the polyline type, width, and color previously associated with this polyline bundle table index.

line_type

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument specifies the type of the polyline. The argument can be any of the following values or constants:

Value	Constant	Description
< 0		Device-dependent types.
1	GKS$K_LINETYPE_SOLID	Use solid line.
2	GKS$K_LINETYPE_DASHED	Use dashed line.
3	GKS$K_LINETYPE_DOTTED	Use dotted line.
4	GKS$K_LINETYPE_DASHED_DOTTED	Use dashed-dotted line.
> = 5		Reserved: future standardization.

See GKS$SET_PLINE_LINETYPE in this section for more information.

line_width

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument is the line width scale factor that is multiplied by the workstation's nominal line width to adjust the width of the polyline. See GKS$SET_PLINE_LINEWIDTH in this section for more information.

color_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the color index of the polyline. See GKS$SET_PLINE_COLOR_INDEX for more information.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
60	GKS$_ERROR_60	Polyline index is invalid in routine ****
63	GKS$_ERROR_63	Specified linetype is equal to zero in routine ****

Error Number	Completion Status Code	Message
64	GKS$_ERROR_64	Specified linetype is not supported on this workstation in routine ****
65	GKS$_ERROR_65	Linewidth scale factor is less than zero in routine ****
93	GKS$_ERROR_93	Color index is invalid in routine ****

Program Example

Example 6–24 illustrates the use of the function GKS$SET_PLINE_REP. Following the program example, Figure 6–28 illustrates the program's effect on a VT241 workstation.

Example 6–24: Changing the Polyline Representation

```
C     This program sets the Attribute Source Flags (ASFs) to bundled,
C     shows the polyline corresponding to the bundle value 8, and then
C     changes the attributes of bundle number 8, using the function
C     GKS$SET_PLINE_REP.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, PLINE_INDEX, GREEN,
     * SEG_NAME
      REAL TIMES_FIVE
      DATA WS_ID / 1 /, NUM_POINTS / 5 /,
     * PLINE_INDEX / 8 /, GREEN / 1 /, TIMES_FIVE / 5.0 /,
     * SEG_NAME / 1 /
      REAL PX( 5 ), PY( 5 )
(1)   DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/
      INTEGER FLAGS( 13 )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
```

(continued on next page)

Example 6–24 (Cont.): Changing the Polyline Representation

```
❷       FLAGS( 1 ) = GKS$K_ASF_BUNDLED
        FLAGS( 2 ) = GKS$K_ASF_BUNDLED
        FLAGS( 3 ) = GKS$K_ASF_BUNDLED
        CALL GKS$SET_ASF( FLAGS )

C       Store the output in a segment.
        CALL GKS$CREATE_SEG( SEG_NAME )
❸       CALL GKS$SET_PLINE_INDEX( PLINE_INDEX )
        CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
        CALL GKS$CLOSE_SEG()

C       Release deferred output. Pause.  Type RETURN when you are finished
C       viewing the picture.
        CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
        READ(5,*)

❹       CALL GKS$SET_PLINE_REP( WS_ID, PLINE_INDEX,
       * GKS$K_LINETYPE_SOLID, TIMES_FIVE, GREEN )

C       Update the screen to reflect the changes.
        CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

        CALL GKS$DEACTIVATE_WS( WS_ID )
        CALL GKS$CLOSE_WS( WS_ID )
        CALL GKS$CLOSE_GKS()
        END
```

The following numbers correspond to the numbers in the previous example:

❶ PX contains the polygon's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ This code initializes the elements of the array that affect all of the non-geometric polyline attributes. This code sets each ASF to GKS$K_ASF_BUNDLED. FLAGS(1) corresponds to the line type; FLAGS(2) corresponds to the line width scale factor; and, FLAGS(3) corresponds to the polyline color index.

See the GKS$SET_ASF function description in this chapter for more information.

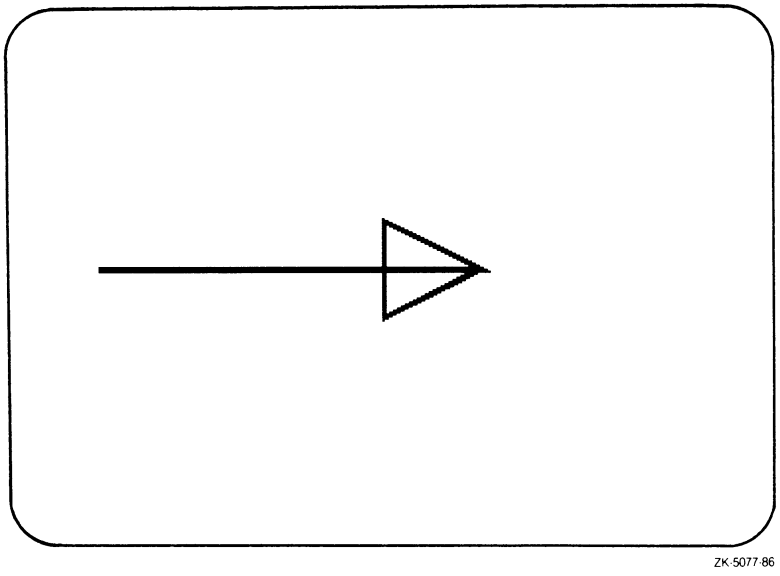
❸ Using the VT241, setting the polyline index to the value 8 generates a red, dotted polyline of nominal width.

❹ On a VT241, calling GKS$SET_PLINE_REP causes an implicit regeneration that is suppressed by the workstation (by default). The attribute changes are not made and the screen is out of date. You need to call GKS$UPDATE_WS to update the surface of the workstation.

If your workstation requires an implicit regeneration to implement changes to polyline representation but does not suppress the regeneration by default, the workstation redraws only the visible segments on the workstation surface. Output primitives not contained in segments are lost. For a complete discussion of implicit regeneration, refer to Chapter 4, Control Functions.

Figure 6–28 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–28: Changing the Polyline Representation—VT241



SET POLYMARKER REPRESENTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_PMARK_REP allows you to redefine an existing polymarker bundle table index representation, or to define a new polymarker bundle table index value, by specifying the marker type, the marker size, and the marker color index associated with the specified bundle index.

Depending on the capabilities of your workstation, a call to this function may cause DEC GKS to implicitly regenerate the workstation surface. For information concerning implicit regeneration, refer to Chapter 4, Control Functions.

Attribute values passed to this function must be valid for the specified workstation. For information, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_PMARK_REP *(workstation_id, polymarker_index, marker_type, marker_size, color_index)*

GSPMR *(workstation_id, pindex, mtype, sfactor, cindex)*

gsetmarkerrep *(workstation_id, index, rep)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in Chapter 4, Control Functions).

polymarker_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the polymarker bundle table index value. By specifying a value to this function, you are redefining the polymarker type, size, and color previously associated with this polymarker bundle table index.

marker_type

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument specifies the type of the polymarker. The argument can be any of the following values or constants:

Value	Constant	Description
< 0		Device-dependent types.
1	GKS$K_MARKERTYPE_DOT	Use dots (.).
2	GKS$K_MARKERTYPE_PLUS	Use plus signs (+).
3	GKS$K_MARKERTYPE_ASTERISK	Use asterisks (*).
4	GKS$K_MARKERTYPE_CIRCLE	Use circles (o).
5	GKS$K_MARKERTYPE_DIAGONAL_CROSS	Use diagonal crosses (X).
> = 6		Reserved: future standardization.

See GKS$SET_PMARK_TYPE in this section for more information.

marker_size

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument is the polymarker size scale factor that is multiplied by the workstation's nominal marker size to adjust the size of the polymarker. See GKS$SET_PMARK_SIZE in this section for more information.

color_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the color index of the polymarker. See GKS$SET_PMARK_COLOR_INDEX for more information.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
66	GKS$_ERROR_66	Polymarker index is invalid in routine ****
69	GKS$_ERROR_69	Marker type is equal to zero in routine ****

Error Number	Completion Status Code	Message
70	GKS$_ERROR_70	Specified marker type is not supported on this workstation in routine ****
71	GKS$_ERROR_71	Specified marker size scale factor is less than zero in routine ****
93	GKS$_ERROR_93	Color index is invalid in routine ****

Program Example

Example 6–25 illustrates the use of the function GKS$SET_PMARK_REP. Following the program example, Figure 6–29 illustrates the program's effect on a VT241 workstation.

Example 6–25: Changing the Polymarker Representation

```
C     This program sets the Attribute Source Flags (ASFs) to bundled,
C     shows the polymarker corresponding to the bundle value 8, and then
C     changes the attributes of bundle number 8, using the function
C     GKS$SET_PMARK_REP.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS,
     * PMARK_INDEX, GREEN, SEG_NAME
      REAL PX( 5 ), PY( 5 ), TIMES_TEN
      DATA WS_ID / 1 /, NUM_POINTS / 5 /,
     * PMARK_INDEX / 8 /, GREEN / 1 /, TIMES_TEN / 10.0 /,
     * SEG_NAME / 1 /
(1)   DATA PX /.1, .9, .7, .7, .9/
      DATA PY /.5, .5, .6, .4, .5/
      INTEGER FLAGS( 13 )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
```

(continued on next page)

Example 6–25 (Cont.): Changing the Polymarker Representation

```
      FLAGS( 4 ) = GKS$K_ASF_BUNDLED
      FLAGS( 5 ) = GKS$K_ASF_BUNDLED
      FLAGS( 6 ) = GKS$K_ASF_BUNDLED
      CALL GKS$SET_ASF( FLAGS )

C     Store the output in a segment.
      CALL GKS$CREATE_SEG( SEG_NAME )
❷     CALL GKS$SET_PMARK_INDEX( PMARK_INDEX )
❸     CALL GKS$POLYMARKER( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

❹     CALL GKS$SET_PMARK_REP( WS_ID, PMARK_INDEX,
     * GKS$K_MARKERTYPE_PLUS, TIMES_TEN, GREEN )

C     Update the surface to reflect the changes.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ PX contains the markers' X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the first point (.1, .5).

❷ This code initializes the elements of the array that affect all of the non-geometric polymarker attributes. This code sets each ASF to GKS$K_ASF_BUNDLED. FLAGS(4) corresponds to the marker type; FLAGS(5) corresponds to the marker size scale factor; and, FLAGS(6) corresponds to the polymarker color index.

See the GKS$SET_ASF function description in this chapter for more information.

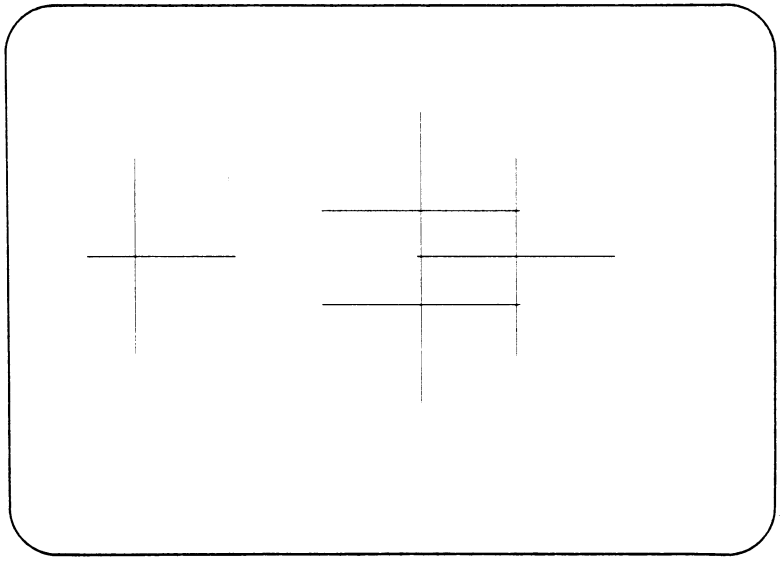
❸ Setting the polymarker index to the value 8 outputs a small, red circle of nominal size.

❹ On a VT241, calling GKS$SET_PMARK_REP causes an implicit regeneration that is suppressed by the workstation (by default). The attribute changes are not made and the screen is out of date. You need to call GKS$UPDATE_WS to update the surface of the workstation.

If your workstation requires an implicit regeneration to implement changes to polymarker representation but does not suppress the regeneration by default, the workstation redraws only the visible segments on the workstation surface. Output primitives not contained in segments are lost. For a complete discussion of implicit regeneration, refer to Chapter 4, Control Functions.

Figure 6–29 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–29: Changing the Polymarker Representation—VT241



ZK-5851-HC

SET TEXT REPRESENTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_TEXT_REP allows you to redefine an existing text bundle table index representation, or to define a new text bundle table index value, by specifying the text font and precision, the character expansion factor, the character spacing, and the text color index associated with the specified bundle index.

Depending on the capabilities of your workstation, a call to this function may cause DEC GKS to implicitly regenerate the workstation surface. For information concerning implicit regeneration, refer to Chapter 4, Control Functions.

Attribute values passed to this function must be valid for the specified workstation. For information, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$SET_TEXT_REP *(workstation_id, text_index, font_value, precision_value, expansion_factor, character_spacing, color_index)*

GSTXR *(workstation_id, tindex, font, precision, efactor, spacing, cindex)*

gsettextrep *(workstation_id, index, rep)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that identifies an open workstation (refer to GKS$OPEN_WS in Chapter 4, Control Functions).

text_index

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the text index value. By specifying a value to this function, you are redefining the associated text font, precision, expansion factor, spacing, and color previously associated with this text index value.

font_value

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the font value. If you are using the character or string precisions, refer to the *DEC GKS Device Specifics Reference Manual.* for more information concerning hardware fonts. If using the stroke precision, refer to Appendix G, DEC GKS Device-Independent Fonts, in this manual.

See GKS$SET_TEXT_FONTPREC in this section for more information concerning the differences between hardware and software fonts.

precision_value

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the precision value. The argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_TEXT_PRECISION_STRING	Lowest precision
1	GKS$K_TEXT_PRECISION_CHAR	Moderate precision
2	GKS$K_TEXT_PRECISION_STROKE	Highest precision

Depending on the precision you choose, you may have to use either hardware or software fonts. See GKS$SET_TEXT_FONTPREC for more information.

expansion_factor

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument is the character expansion factor. This value multiplied by the width-to-height ratio specified in the original font specification determines the new character width. The character height remains the same.

character_spacing

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument is the spacing factor. This value, multiplied times the text height, is the spacing value in world coordinates. If you specify a positive number, the spacing increases between letters (for example, the value 0.1 sets spacing to be one tenth the character height). If you specify a negative number, the spacing decreases (characters may overlap). If you specify the value 0.0, the bodies of the characters are adjacent, without any separating space not defined as part of the character body by the font design.

See GKS$SET_TEXT_SPACING in this section for more information.

color_index

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the color index of the text. See GKS$SET_TEXT_COLOR_INDEX for more information.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
72	GKS$_ERROR_72	Text index is invalid in routine ****
75	GKS$_ERROR_75	Text font is equal to zero in routine ****

Error Number	Completion Status Code	Message
76	GKS$_ERROR_76	Requested text font is not supported for the specified precision on this workstation
77	GKS$_ERROR_77	Character expansion factor is less than or equal to zero in routine ****
93	GKS$_ERROR_93	Color index is invalid in routine ****

Program Example

Example 6–26 illustrates the use of the function GKS$SET_TEXT_REP. Following the program example, Figure 6–30 illustrates the program's effect on a VT241 workstation.

Example 6–26: Changing the Text Representation

```
C     This program changes the text representation of the index
C     value 5.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, FIVE, OLD_ENGLISH, GREEN, SEG_NAME
      REAL LARGER, START_PT_X, START_PT_Y, ONE_TENTH, TIMES_TWO
      DATA WS_ID / 1 /, LARGER / 0.03 /, FIVE / 5 /,
     * START_PT_X / 0.1 /, START_PT_Y / 0.5 /,
     * OLD_ENGLISH / -18 /, ONE_TENTH / 0.1 /, GREEN / 1 /,
     * TIMES_TWO / 0.02 /, SEG_NAME / 1 /
      INTEGER FLAGS( 13 )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

(1)   CALL GKS$SET_TEXT_HEIGHT( LARGER )
```

(continued on next page)

Example 6–26 (Cont.): Changing the Text Representation

```
❷       FLAGS( 7 ) = GKS$K_ASF_BUNDLED
        FLAGS( 8 ) = GKS$K_ASF_BUNDLED
        FLAGS( 9 ) = GKS$K_ASF_BUNDLED
        FLAGS( 10 ) = GKS$K_ASF_BUNDLED
        CALL GKS$SET_ASF( FLAGS )

C       Store output in a segment.
        CALL GKS$CREATE_SEG( SEG_NAME )
❸       CALL GKS$SET_TEXT_INDEX( FIVE )
        CALL GKS$TEXT( START_PT_X, START_PT_Y, 'Imitation Life' )
        CALL GKS$CLOSE_SEG()

C       Release deferred output. Pause.  Type RETURN when you are finished
C       viewing the picture.
        CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
        READ(5,*)

❹       CALL GKS$SET_TEXT_REP( WS_ID, FIVE, OLD_ENGLISH,
       * GKS$K_TEXT_PRECISION_STROKE, TIMES_TWO, ONE_TENTH, GREEN )

C       Update the surface to reflect the changes.
        CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

        CALL GKS$DEACTIVATE_WS( WS_ID )
        CALL GKS$CLOSE_WS( WS_ID )
        CALL GKS$CLOSE_GKS()
        END
```

The following numbers correspond to the numbers in the previous example:

❶ This code increases the character height so that the string is easy to see.

❷ This code initializes the elements of the array that affect all of the nongeometric polymarker attributes. This code sets each ASF to GKS$K_ASF_BUNDLED. FLAGS(7) corresponds to the text font and precision; FLAGS(8) corresponds to the character expansion factor; FLAGS(9) corresponds to the character spacing; and, FLAGS(10) corresponds to the text color index.

See the GKS$SET_ASF function description in this chapter for more information.

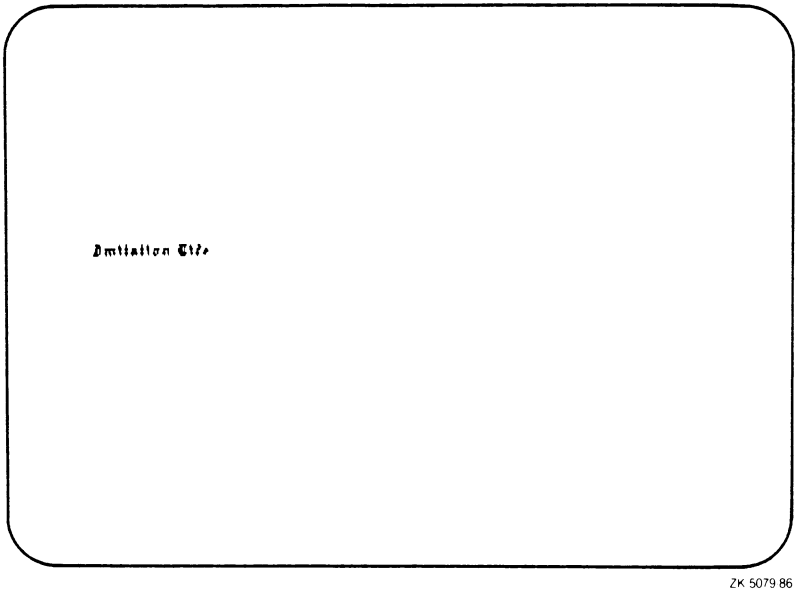
❸ Setting the text index to the value 5 outputs red text in stroke precision, with hardware font 1 at the default width and spacing.

❹ On a VT241, calling GKS$SET_TEXT_REP causes an implicit regeneration that is suppressed by the workstation (by default). The attribute changes are not made and the screen is out of date. You need to call GKS$UPDATE_WS to update the surface of the workstation.

If your workstation requires an implicit regeneration to implement changes to text representation but does not suppress the regeneration by default, the workstation redraws only the visible segments on the workstation surface. Output primitives not contained in segments are lost. For a complete discussion of implicit regeneration, refer to Chapter 4, Control Functions.

Figure 6–30 shows the screen of a VT241 terminal after the program has run to completion.

Figure 6–30: Changing the Text Representation—VT241



Chapter 7

Transformation Functions

The DEC GKS transformation functions allow you to compose a picture, to control how much of the picture is seen on the workstation surface, and to control how much of the workstation surface is used to display the picture. The following list presents the transformation functions by category:

Category	GKS Functions
Normalization	GKS$SELECT_XFORM, GKS$SET_CLIPPING, GKS$SET_VIEWPORT, GKS$SET_VIEWPORT_PRIORITY, GKS$SET_WINDOW
Workstation	GKS$SET_WS_VIEWPORT, GKS$SET_WS_WINDOW

When you request input and generate output by means of the workstation surface, you are actually working with a number of coordinate systems. The image is *transformed* from one coordinate system to the next.

Using DEC GKS, you are working with three coordinate systems, as follows:

- World coordinate system
- Normalized device coordinate (NDC) system
- Device coordinate system

The world coordinate system is an imaginary coordinate plane used to plot a graphical image. The NDC system is a device-independent, imaginary coordinate plane on which you compose a picture using designated portions of the world coordinate plane. Once you compose a picture on the NDC space, you can zoom in on the picture, pan across the picture, or zoom out of the picture, while controlling what portion of the device coordinate system is used to display the picture. You can display all or part of the picture in NDC space on the surface of the physical device.

When you call one of the DEC GKS output functions, you specify world coordinate points. Using a series of default windows and viewports, the output primitive is transformed from an image on the world coordinate plane, to an image on the NDC plane, and finally, to the surface of the workstation.

If you do not change the default transformation settings, image shape and position are consistent, and your ability to compose complex pictures may be limited to what you can form on one area of the world coordinate system. The DEC GKS transformation functions allow you to set the windows, viewports, and other transformation features that control the transformation process, and usually, how generated output appears on the workstation surface.

7.1 World Coordinates and Normalization Transformations

The world coordinate system is an imaginary, Cartesian coordinate system whose X and Y axes extend infinitely in all four directions. The origin of the system is the point (0.0, 0.0). Depending on the type of data needed to plot your images, you can use any portion of the world coordinate plane. For instance, if the necessary data contains negative numbers, you can use the portions of the world coordinate system that extend into the negative portions of the axes.

By default, DEC GKS transforms images according to a square world coordinate range whose lower left corner is the point (0.0, 0.0) and whose sides extend from the point 0.0 to 1.0 on both the X and Y axes. (From this point forward, this manual documents rectangular regions such as the default range as follows: ([0,1] x [0,1]), zero to one on both the X and Y axes.) This range is called the default normalization *window*.

DEC GKS transforms the plotted images, according to the current window, to an area on the NDC plane. You can reset the window many times while generating output primitives, or you can use only the default window, depending on the needs of your application. If your image is composed of points that lie outside of the world window, then those points may or may not be part of the image on the NDC plane depending on the current *clipping* indicator. Clipping is discussed in detail in Section 7.1.1.

As an example, consider the formation of a picture of a house on the world coordinate plane. To illustrate the resetting of the normalization window, consider a coordinate range of ([0,10] x [0,10]). The following code example shows how to set such a range for the normalization window.

```
.
.
.
DATA PX / 4.0, 1.0, 1.0, 4.0, 2.5, 1.0, 4.0, 4.0, 1.0 /
DATA PY / 1.0, 1.0, 7.0, 7.0, 9.0, 7.0, 1.0, 7.0, 1.0 /
CALL GKS$SET_WINDOW( 1, 0.0, 10.0, 0.0, 10.0 )
CALL GKS$SELECT_XFORM( 1 )
CALL GKS$POLYLINE( 9, PX, PY )
```

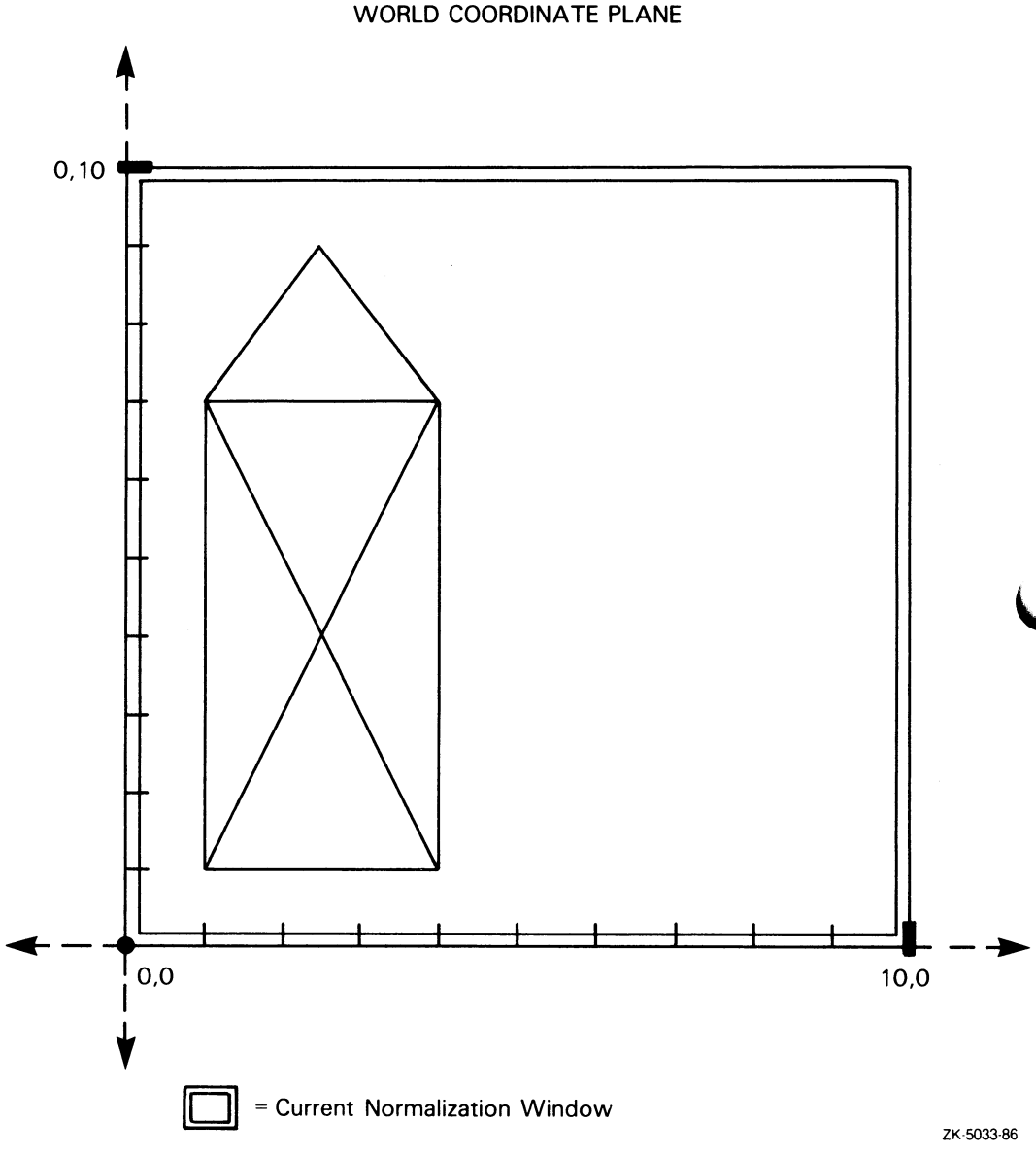
PX contains the house's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the lower right corner of the house (4.0, 1.0).

In the call to GKS$SET_WINDOW, the window's X axis minimum value is set to the point 0.0 and its maximum value is set to 10.0. The window's Y axis minimum and maximum values are set to the same world coordinates. These dimensions establish the rectangle used as the normalization window.

The first argument to GKS$SET_WINDOW (the number 1) specifies a *normalization transformation* number. A normalization transformation is a transposition of an image from the world coordinate plane to the NDC plane. When you select normalization transformation number 1 by calling GKS$SELECT_XFORM, DEC GKS establishes a window of the range ([0,10] x [0,10]) to be the current normalization window. When you generate output using this code example, DEC GKS maps the current window to a default portion of the NDC space. Section 7.1.1 describes the NDC plane in detail.

Figure 7–1 illustrates the formation of the house on the world coordinate plane.

Figure 7–1: The World Coordinate Plane



7.1.1 The Normalized Device Coordinate (NDC) System

As mentioned in the previous section, the normalization transformation is the transposition of world coordinate points to NDC points. The NDC plane is a device-independent coordinate plane on which you compose graphical pictures. The NDC plane has an X and Y axis that in theory extends infinitely in all four directions with an origin at point (0.0, 0.0), but in practice, only images contained in the range ([0,1] x [0,1]) can ultimately be transformed to the surface of a physical device.

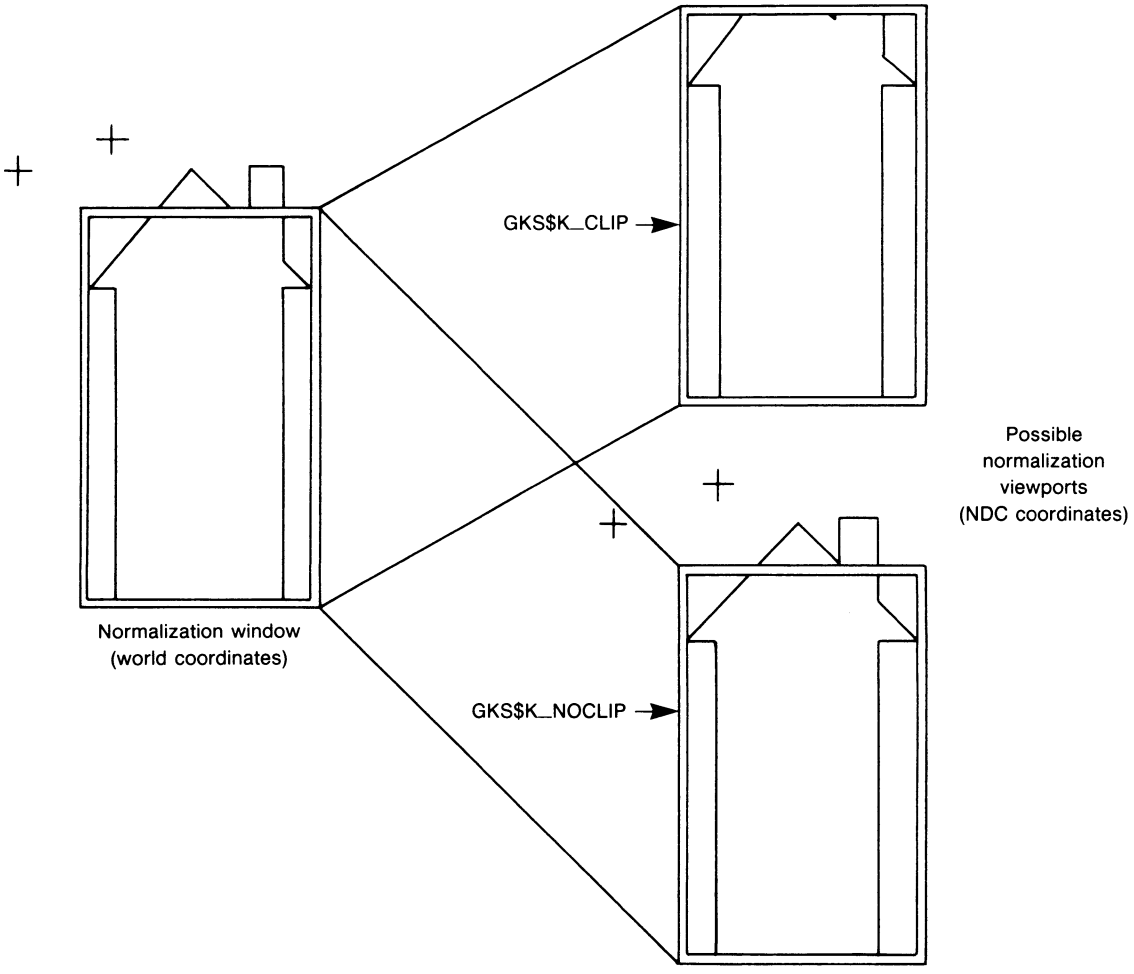
When DEC GKS transforms an image from the normalization window to the NDC plane, there must be a corresponding rectangle on which to map the contents of the window. This rectangular portion of the NDC space is called the normalization *viewport*. The default viewport has the range ([0,1] x [0,1]) in NDC point values. The previous code example, by default, maps the contents of the current window to this default viewport.

By default, DEC GKS maps the normalization window ([0,1] x [0,1]) in world coordinates to the viewport ([0,1] x [0,1]) in NDC point values. This transformation is called the *unity* transformation, which has the normalization transformation number 0. You cannot reset the window and viewport associated with the unity transformation. All of the examples in Chapter 5, Output Functions, use the unity transformation.

You can think of the normalization process as a way of possibly transposing a number of areas of the world coordinate plane onto the NDC plane in respect to current normalization window and viewport. For instance, DEC GKS maps the contents of the current normalization window onto the current viewport. If clipping is enabled (which is the default situation), the effect is like cutting the window from the world coordinate plane, mapping, and then pasting the window to the viewport on the NDC plane. DEC GKS maps only images or portions of images plotted within the boundaries of the normalization window to the area within the viewport on NDC space. If clipping is *disabled*, DEC GKS maps points that lie outside of the normalization window boundary to NDC space outside of the normalization viewport but within the range ([0,1] x [0,1]).

Since DEC GKS clips images at the boundary of the normalization viewport, this viewport is also called the *clipping rectangle*. You can enable and disable clipping by calling the function GKS$SET_CLIPPING. Figure 7–2 illustrates the clipping process according to the argument passed to GKS$SET_CLIPPING.

Figure 7–2: The Clipping Rectangle



As one option to consider while creating a picture, you can select different normalization transformations with different windows and viewports, thus mapping various portions of the world coordinate space onto different portions of the NDC space. (In DEC GKS, valid normalization transformation numbers range from 0 to 255, and can associate windows and viewports with all but the unity transformation number 0.) You can achieve the same effect by reassigning different windows and viewports to a single normalization number.

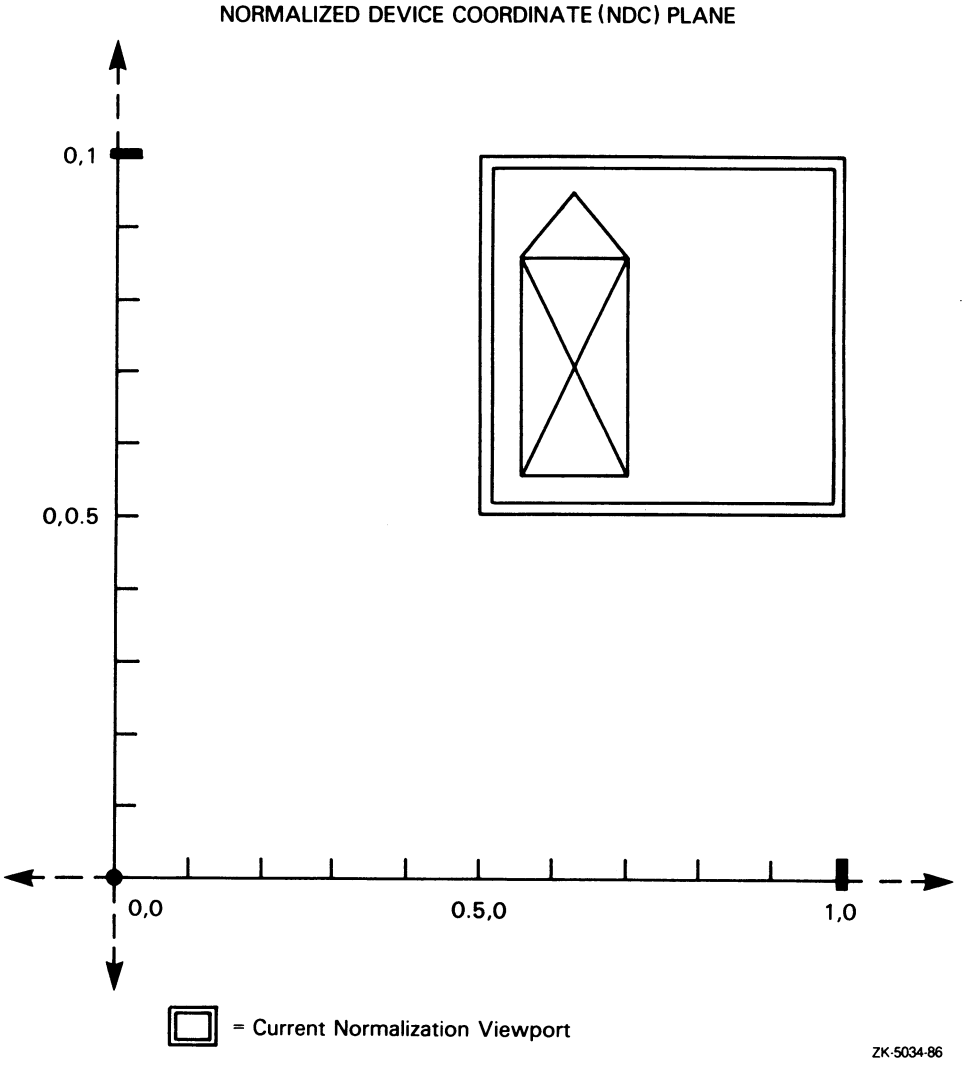
In essence, you use the world coordinate space as a scratch pad and the NDC space as a pasteboard on which to compose an entire picture. For instance, if you want an output primitive to appear on the right side of a picture appearing on the workstation surface, you map the primitive to the right side of the *NDC space* during the normalization transformation. All picture composition is done using normalization transformations. Once you compose a picture on the NDC plane, you can output all or part of the picture to all or part of various workstation surfaces.

The following code examples show how to compose a picture using different normalization windows and viewports.

```
      .
      .
      .
DATA PX / 4.0, 1.0, 1.0, 4.0, 2.5, 1.0, 4.0, 4.0, 1.0 /
DATA PY / 1.0, 1.0, 7.0, 7.0, 9.0, 7.0, 1.0, 7.0, 1.0 /

CALL GKS$SET_WINDOW( 1, 0.0, 10.0, 0.0, 10.0 )
CALL GKS$SET_VIEWPORT( 1, 0.5, 1.0, 0.5, 1.0 )
CALL GKS$SELECT_XFORM( 1 )

CALL GKS$POLYLINE( 9, PX, PY )
```

Normalization transformation number 1 transforms the window in Figure 7–1 to a portion of the NDC space as shown in Figure 7–3.

Figure 7–3: The Normalization Viewport

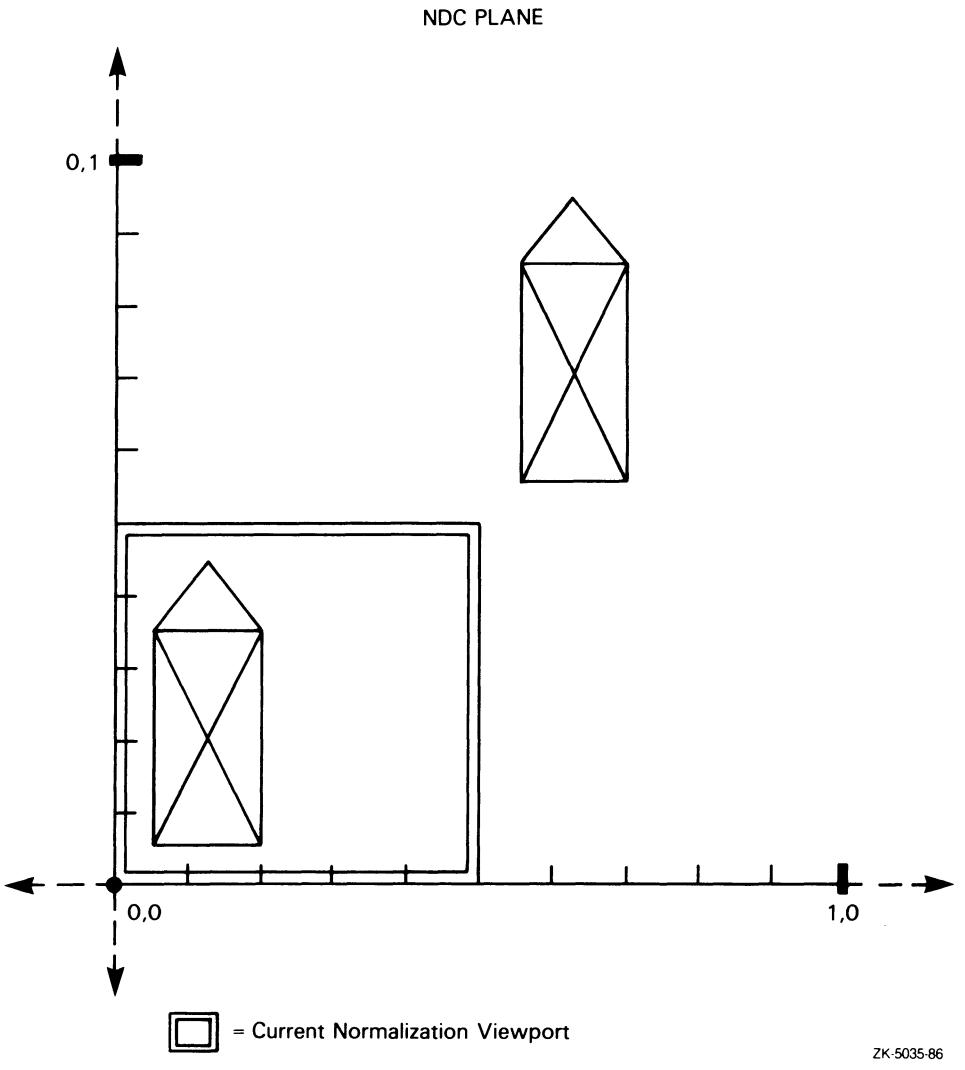


By selecting a different normalization transformation with a different viewport, you can transpose the same window onto another portion of the NDC space. To see how this is accomplished, review the following code example:

```
      .
      .
      .
      DATA PX / 4.0, 1.0, 1.0, 4.0, 2.5, 1.0, 4.0, 4.0, 1.0 /
      DATA PY / 1.0, 1.0, 7.0, 7.0, 9.0, 7.0, 1.0, 7.0, 1.0 /

      CALL GKS$SET_WINDOW( 1, 0.0, 10.0, 0.0, 10.0 )
      CALL GKS$SET_VIEWPORT( 1, 0.5, 1.0, 0.5, 1.0 )

      CALL GKS$SET_WINDOW( 2, 0.0, 10.0, 0.0, 10.0 )
      CALL GKS$SET_VIEWPORT( 2, 0.0, 0.5, 0.0, 0.5 )

C     Make normalization transformation #1 the current transformation.
      CALL GKS$SELECT_XFORM( 1 )
      CALL GKS$POLYLINE( 9, PX, PY )

C     Make normalization transformation #2 the current transformation.
      CALL GKS$SELECT_XFORM( 2 )
      CALL GKS$POLYLINE( 9, PX, PY )
```

After execution of this code example, the NDC space contains the images in Figure 7–4. In this manner, you can map a number of normalization windows to a number of viewports until you compose a complete picture on the NDC space. See Section 7.2 to learn how to display the composed picture on the workstation surface.

Figure 7–4: Composing a Picture on the NDC Plane



7.1.2 Overlapping Viewports

When you define normalization viewports, it is possible to cause them to overlap on the NDC plane. You must consider the effects this has during input requests. Viewport input priority does not affect output; the order of the output function calls determines which primitive overwrites the other. If you are working with segments, the segment priorities affect overlapping segments. For more information, refer to Chapter 9, Segment Functions.

To illustrate the need for a viewport priority list for use during input, consider two viewports: the viewport of the unity transformation number 0 ([0,1] x [0,1]), and a viewport, belonging to normalization transformation number 1, having the range ([0.5,1] x [0.5,1]) in NDC points. Notice that the viewport of normalization transformation number 1 overlaps the right side of the unity viewport.

During stroke and locator input, the user positions the cursor on the device surface and returns one point (locator) or a series of points (stroke) in device coordinates. DEC GKS translates the device coordinates to NDC points (Section 7.2 discusses this process in detail).

Once the device coordinates are transformed to NDC points, DEC GKS must transform the NDC points to world coordinate points. To transform the point, DEC GKS transforms the point from its viewport (NDC) value to the corresponding window (world coordinate) value. However, if the user chooses a point on the right half of the default viewport, DEC GKS must decide whether to use the unity viewport or the overlapping viewport of transformation number 1 to transform the point to world coordinates. DEC GKS needs to know to which normalization window the point is to be mapped, either the window that corresponds to normalization transformation number 0, or 1.

To decide which viewport has a higher input priority, DEC GKS maintains a priority list. By default, DEC GKS assigns the highest priority to the unity transformation (0). So, in the previous example concerning overlapping viewports, DEC GKS would use the unity viewport to transform the NDC point. The viewports of all remaining transformations decrease in priority as their transformation numbers increase (viewport 0 higher than viewport 1, 1 higher than 2, 2 higher than 3, and so forth).

If you want to change the order of the viewport input priority list, you must call the function GKS$SET_VIEWPORT_PRIORITY. You specify a normalization transformation number whose priority is to be changed (for example, 1), a normalization transformation number as a reference (for example, 0), and a flag that specifies that the first transformation is to have a lower or higher priority than the reference transformation (for example, higher).

So, if you called GKS$SET_VIEWPORT_PRIORITY to give transformation number 1 a higher transformation (1 higher than 0, 0 higher than 2, 2 higher than 3, and so forth), then DEC GKS would use the viewport corresponding to transformation number 1 in all cases when viewports 1 and 0 overlap during locator and stroke input.

For more information concerning locator and stroke input, refer to Chapter 8, Input Functions.

7.2 Workstation Transformations

DEC GKS must map the picture on the NDC plane to the surface of one or more workstations. To do this, DEC GKS uses a second window and a viewport called the *workstation window* and the *workstation viewport*. The workstation window is a rectangular portion of the NDC plane that is mapped onto the rectangular portion of the workstation surface called the workstation viewport. Whereas there can be numerous normalization transformations, there is only one current workstation window and one current workstation viewport.

DEC GKS uses a default workstation window of the range ([0,1] x [0,1]) in NDC points, and uses a default workstation viewport, starting at the lower left corner, that is the largest rectangle on the device surface that maintains the shape of the picture in the workstation window (Section 7.3 discusses in detail the shape of the picture in the workstation window). If you choose, you can change the workstation window, but the new boundaries can be no larger than the default workstation window boundaries ([0,1] x [0,1]). DEC GKS clips all points that exceed the default workstation window boundaries before DEC GKS transforms the picture to device coordinates, regardless of the current clipping flag setting.

Whereas the normalization transformation composes the picture on NDC space, the workstation transformation *presents* all or part of the picture on all or part of the device surface. For instance, by setting the workstation window, you can create the illusion of "panning" across a picture, showing successive portions of it at a time, or "zooming in," showing smaller portions of a picture at a time. The *DEC GKS User Manual* discusses this process in detail.

Your application may require that you change the portion of the workstation surface used to display the picture. However, if your program runs on several devices, you may not know the proportions of the device coordinate system with which you are working. The proportions of the device coordinate system are completely device dependent; each device can have a completely dissimilar device coordinate plane with dissimilar maximum X and Y coordinate values.

To determine the maximum boundary of the workstation viewport, you should use the function, GKS$INQ_MAX_DS_SIZE, which returns the maximum X and Y values of the workstation display surface. (For more information, refer to Chapter 12, Inquiry Functions, or to GKS$SET_WS_VIEWPORT in this chapter.)

When you set the workstation window (by calling GKS$SET_WS_WINDOW) or the workstation viewport (by calling GKS$SET_WS_VIEWPORT), the new window or viewport may not come into effect immediately, depending on the capabilities of your device. Depending on your device, the new workstation window or workstation viewport may become current immediately, or the workstation surface may need to be implicitly regenerated before the new window or viewport becomes current. If the workstation needs to regenerate its surface to make a workstation transformation current, the screen is cleared and only the primitives stored in segments are redrawn. You lose all primitives not contained in segments.

For a detailed discussion of implicit regeneration and surface update, refer to Chapter 4, Control Functions.

The following code example shows how to change the workstation window and viewport on a device that suppresses implicit regenerations:

```
      .
      .
      .
      CALL GKS$SET_WINDOW( 1, 0.0, 10.0, 0.0, 10.0 )
      CALL GKS$SET_VIEWPORT( 1, 0.5, 1.0, 0.5, 1.0 )
      CALL GKS$SET_WINDOW( 2, 0.0, 10.0, 0.0, 10.0 )
      CALL GKS$SET_VIEWPORT( 2, 0.0, 0.5, 0.0, 0.5 )

C     Find the maximum X and Y device coordinate values of your
C     device's type.
      CALL GKS$INQ_MAX_DS_SIZE( GKS$K_VT240, ARG2, ARG3, MAX_X,
     * MAX_Y, ARG6, ARG7 )

C     Set a new workstation window in NDC points.
      CALL GKS$SET_WS_WINDOW( 1, 0.0, 1.0, 0.25, 1.0 )
      CALL GKS$SET_WS_VIEWPORT( 1, 0.0, MAX_X, 0.0, MAX_Y )

C     Update the screen.  Primitives not stored in segments are lost.
      CALL GKS$UPDATE_WS( 1, GKS$K_PERFORM_FLAG )

C     Make normalization transformation #1 the current transformation.
      CALL GKS$SELECT_XFORM( 1 )
      CALL GKS$POLYLINE( 9, PX, PY )

C     Make normalization transformation #2 the current transformation.
      CALL GKS$SELECT_XFORM( 2 )
      CALL GKS$POLYLINE( 9, PX, PY )
```

After execution of this code example, the NDC space contains the images in Figure 7–5. Depending on your device, the surface of your workstation would look like Figure 7–6. In most instances, the workstation does not use the entire workstation viewport to display the picture. DEC GKS uses the portion of device coordinate space, starting at the lower left point, that is the largest rectangle within the current workstation viewport that maintains the shape of the picture contained in the workstation window. In order to map the entire workstation window to the entire viewport, you need to make sure that the window and viewport have the same proportions. See Section 7.3 for more information concerning window and viewport proportions. The *DEC GKS User Manual* contains examples working with the proportions of workstation windows and viewports.

The entire process of an image generation, from normalization transformation to workstation transformation, is illustrated in Figure 7–7.

Figure 7–5: The Workstation Window

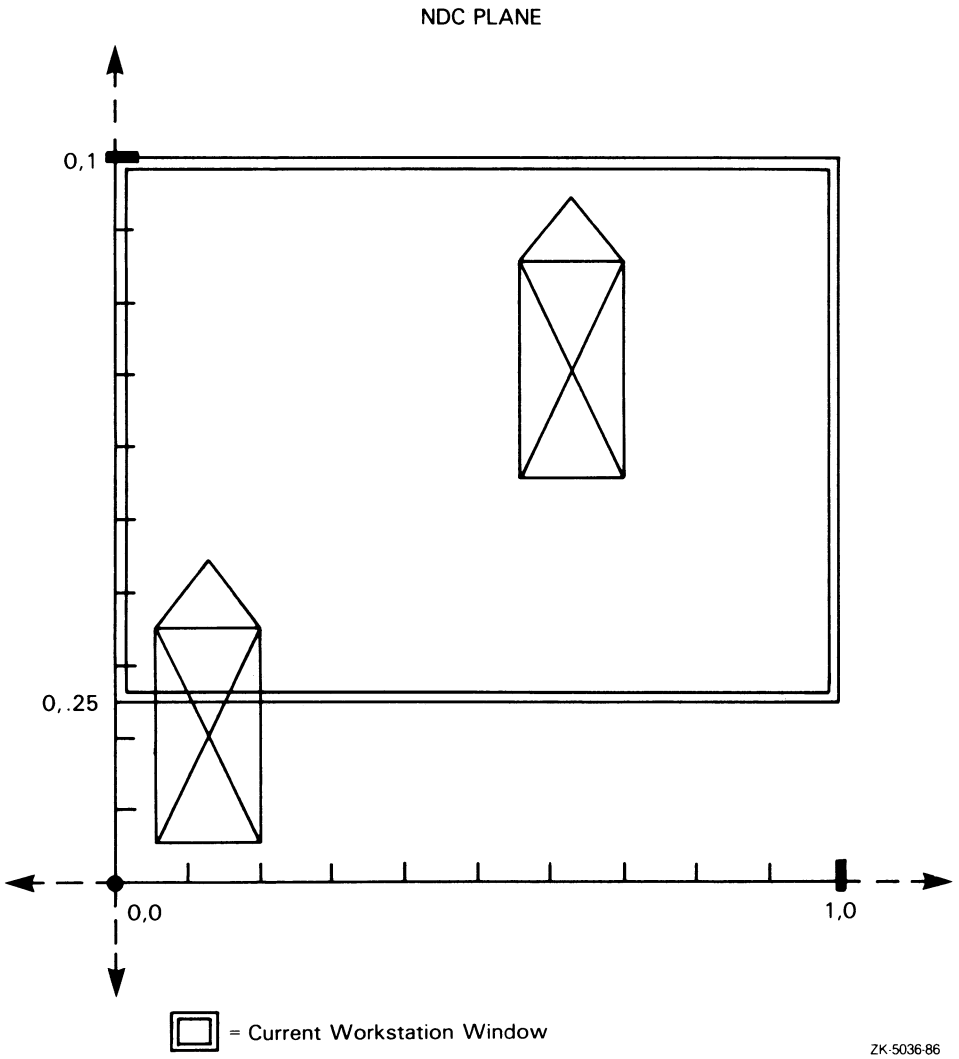


Figure 7–6: The Picture on a Generic Device Surface

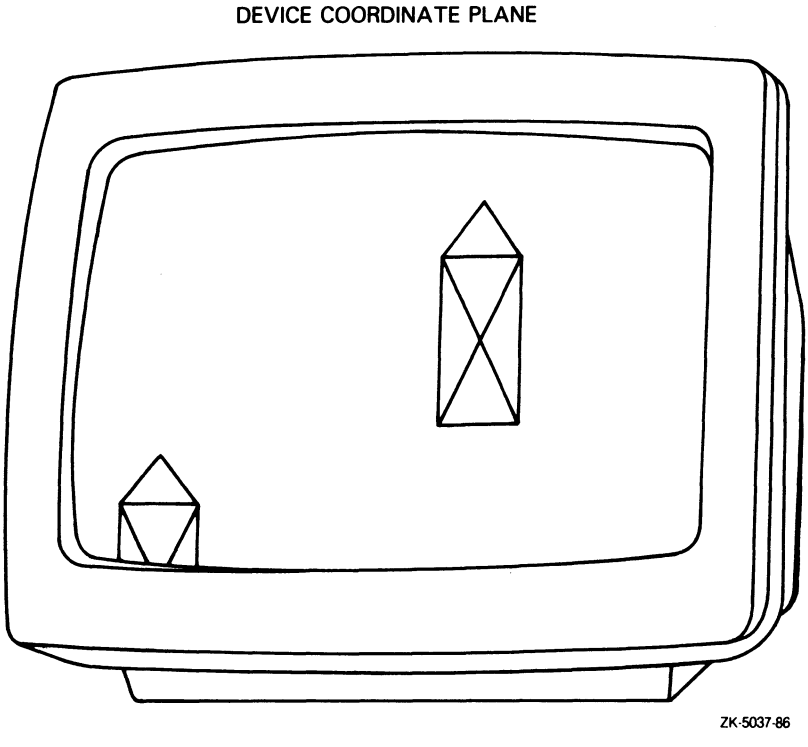
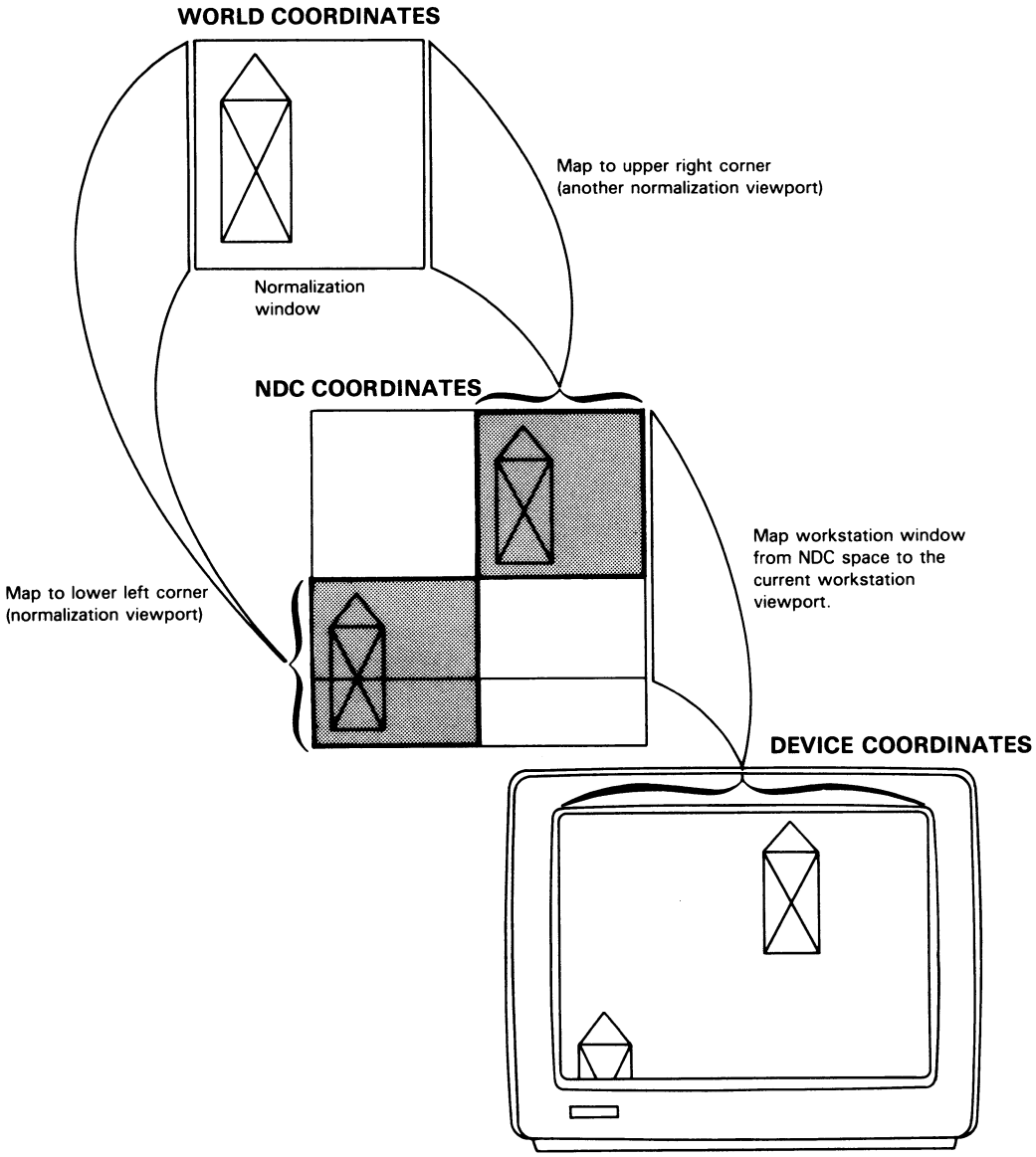


Figure 7–7: The Entire DEC GKS Transformation Process



7.3 Relative Positioning and Shape

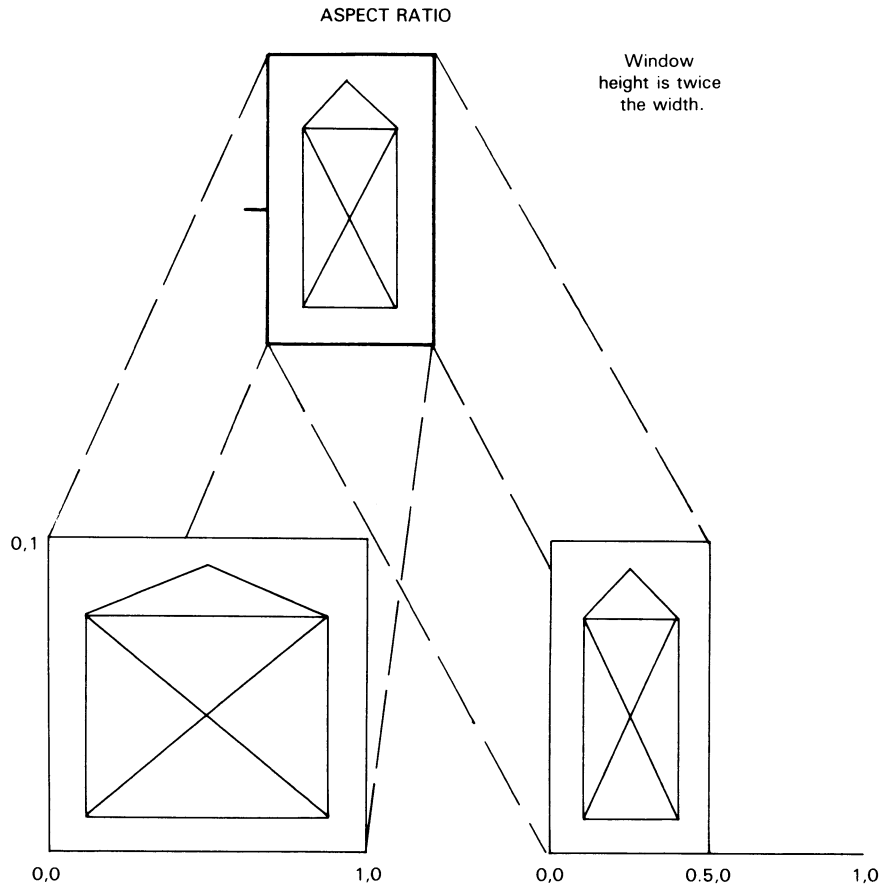
There is one final consideration when redefining the normalization and workstation window and viewport values, and that is the shape of the object to be drawn. By default, the image is mapped from a square world coordinate plane, to a square portion of the NDC plane, and finally to a square portion of the display surface plane. Logically, if you draw a tall, thin house in world coordinate values using the default transformations, you would see a tall, thin house on the workstation surface. However, if you define a tall, thin rectangular *normalization window* that contains your house and then map that window onto the default, square normalization viewport, the tall, thin house would appear shorter and wider due to mapping from window to viewport.

Consequently, you should be aware of the difference between the *relative position* of the image and the *aspect ratio* of the image. In the case of the tall, thin house, all of the points retain their relative position when mapped from the normalization window to the viewport. If the X value of the tip of the house is located two-thirds of the way along the X axis in the window, it is also located two-thirds of the way along the X axis in the viewport. Relatively speaking, if the house is located in the center of the window, it will be located in the center of the viewport.

If you want to retain the shape, or aspect ratio, of the image, you must map images from a window to a viewport that has the same proportion, or height-to-width ratio. For example, if the X axis is two-thirds as large as the Y axis in the normalization window, you must map to a viewport whose X axis is two-thirds as large as the Y axis in order to retain the aspect ratio of your image.

Figure 7–8 shows the difference between relative position and aspect ratio. Like the window and viewport on the left, all normalization transformations retain the relative position. The window and viewport on the right retain the aspect ratio of the tall, thin house as well as its relative position.

Figure 7–8: Relative Position and Aspect Ratio



In contrast to the normalization transformations, DEC GKS automatically retains the aspect ratio of the workstation window when mapping to the workstation viewport. The mapping from workstation window to workstation viewport is not necessarily one-to-one; DEC GKS might not use the entire defined workstation viewport. By default, DEC GKS uses the largest rectangle, starting at the lower left corner, *within* the workstation viewport that retains the shape of the picture contained in the workstation window.

For more information concerning normalization transformations, workstation transformations, relative positioning, and aspect ratio, refer to the *DEC GKS User Manual*. For more information concerning segments and transformations, refer to Chapter 9, Segment Functions.

7.4 Transformation Inquiries

The following list presents the inquiry functions that you can use to obtain transformation information when writing device-independent code:

GKS$INQ_CLIP
GKS$INQ_CURRENT_XFORMNO
GKS$INQ_MAX_DS_SIZE
GKS$INQ_MAX_XFORM
GKS$INQ_WS_XFORM
GKS$INQ_XFORM
GKS$INQ_XFORM_LIST

For more information concerning device-independent programming, refer to the *DEC GKS User Manual*.

7.5 Function Descriptions

This section describes the DEC GKS transformation functions in detail.

SELECT NORMALIZATION TRANSFORMATION

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SELECT_XFORM sets the normalization transformation number in the DEC GKS state list as the current transformation, and uses the associated window and viewport to transform points from the world coordinate system to the NDC system for subsequent output generation.

By default, DEC GKS uses the unity normalization transformation number 0. Use this when you want to map the default normalization window to the default NDC viewport.

Syntax

GKS$SELECT_XFORM *(transformation_number)*

GSELNT *(xform)*

gselntran *(transform)*

Arguments

transformation_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the normalization transformation number. To set or reset windows and viewports associated with a transformation number, you pass this number to GKS$SET_WINDOW and GKS$SET_VIEWPORT. After selecting this number, any subsequent calls to output functions use the window and viewport associated with this number.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC or SGOP in routine ****
50	GKS$_ERROR_50	Transformation number is invalid in routine ****

Program Example

Example 7–1 illustrates the use of the function GKS$SELECT_XFORM. Following the program example, Figure 7–9 illustrates the program's effect on a VT241 workstation.

Example 7–1: Selecting a Normalization Transformation

```
C     This program changes the world viewport four times placing
C     the "tall, thin house" in each corner of the NDC space.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, UP_LEFT_CORNER, UP_RIGHT_CORNER,
     * LOW_LEFT_CORNER, LOW_RIGHT_CORNER, NUM_POINTS
      REAL PX ( 9 ), PY ( 9 ), PX_2 ( 5 ), PY_2 ( 5 )
❶     DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA PX_2 / 0.0, 1.0, 1.0, 0.0, 0.0 /
      DATA PY_2 / 0.0, 0.0, 1.0, 1.0, 0.0 /
      DATA UP_LEFT_CORNER / 1 /, UP_RIGHT_CORNER / 2 /,
     * LOW_LEFT_CORNER / 3 /, LOW_RIGHT_CORNER / 4 /,
     * NUM_POINTS / 9 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❷     CALL GKS$SET_VIEWPORT( UP_LEFT_CORNER, 0.0, 0.5, 0.5, 1.0)
      CALL GKS$SET_VIEWPORT( UP_RIGHT_CORNER, 0.5, 1.0, 0.5, 1.0)
      CALL GKS$SET_VIEWPORT( LOW_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5)
      CALL GKS$SET_VIEWPORT( LOW_RIGHT_CORNER, 0.5, 1.0, 0.0, 0.5)

C     Outlining the default world window will result in the outlining of
C     the NDC plane, the workstation window, and the workstation
C     viewport, by default.
      CALL GKS$POLYLINE( 5, PX_2, PY_2 )

❸     CALL GKS$SELECT_XFORM( UP_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the screen.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
      CALL GKS$SELECT_XFORM( UP_RIGHT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the screen.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
```

(continued on next page)

Example 7–1 (Cont.): Selecting a Normalization Transformation

```
      CALL GKS$SELECT_XFORM( LOW_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the screen.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
      CALL GKS$SELECT_XFORM( LOW_RIGHT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ PX contains the house's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the lower right corner of the house (0.4, 0.1).

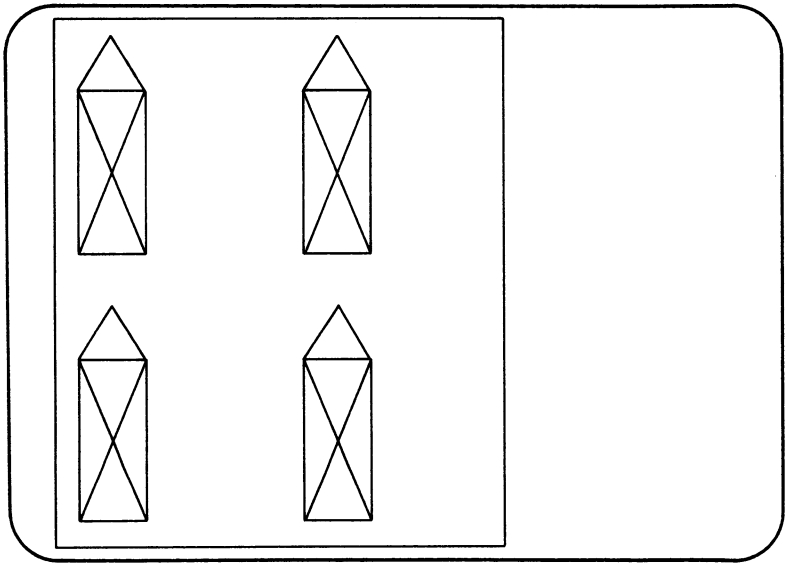
❷ This code sets the viewport to be the upper left corner of the NDC space and assigns this normalization transformation the value 1. The subsequent lines of code set the viewport to be the other three corners of the NDC space and assign the normalization transformations the values 2, 3, and 4.

❸ This code selects the normalization transformation number 1, which corresponds to a default window and to a viewport that is the upper left corner of the NDC space. Until another transformation number is selected, output is mapped to the upper left corner of the default NDC plane. In this example, the tall, thin house is output to the upper left corner of the workstation surface. The next four sections of code change the world viewports to the other three corners of the NDC space and then generate the tall, thin house in the remaining corners of the default NDC space.

When DEC GKS maps the default workstation window to the workstation surface, DEC GKS maps the window to the largest square area on the workstation surface (since the workstation window is square) so as to maintain the shape of the picture.

Figure 7–9 shows the screen of a VT241 terminal after the program has run to completion.

Figure 7–9: Selecting the Normalization Transformation—VT241



ZK-5080-86

SET CLIPPING INDICATOR

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_CLIPPING enables or disables clipping of the image at the normalization viewport boundary by setting the clipping flag in the DEC GKS state list.

If clipping is enabled, DEC GKS clips all generated output primitives at the normalization viewport boundary. If clipping is disabled, primitives may exceed the normalization viewport boundaries. By default, DEC GKS clips primitives.

NOTE

This function works only for the normalization viewport. Pictures are always clipped at the workstation window despite the current status of the clipping flag.

Syntax

GKS$SET_CLIPPING *(clip)*

GSCLIP *(flag)*

gsetclip *(indicator)*

Arguments

clip

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument determines whether or not clipping is enabled or disabled. The argument can be either of the following values or constants.

Value	Constant	Description
0	GKS$K_NOCLIP	Clipping is off.
1	GKS$K_CLIP	Clipping is on.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-23	DECGKS$_ERROR_NEG_23	Invalid value specified for clipping flag in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC or SGOP in routine ****

Program Example

Example 7–2 illustrates the use of the function GKS$SET_CLIPPING. Following the program example, Figure 7–10 illustrates the program's effect on a VT241 workstation.

Example 7–2: Controlling Clipping at the World Viewport

```
C     This program generates a "tall, thin house" that overlaps
C     the normalization window and viewport.  You can see
C     the overlapping portion if clipping is disabled.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, HALF, LOW_LEFT_CORNER, NUM_POINTS
      REAL PX ( 9 ), PY ( 9 ), PX_2 ( 5 ), PY_2 ( 5 )
(1)   DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA PX_2 / 0.0, 0.5, 0.5, 0.0, 0.0 /
      DATA PY_2 / 0.0, 0.0, 0.5, 0.5, 0.0 /
      DATA NUM_POINTS / 9 /, HALF / 1 /,
     * LOW_LEFT_CORNER / 1 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Outlining the default world window will result in the outlining of
C     the NDC plane, the workstation window, and the workstation
C     viewport, by default.
      CALL GKS$POLYLINE( 5, PX_2, PY_2 )

C     This window (half of the default window) and viewport (lower
C     left corner of the default viewport) are associated with
C     normalization transformation number 1.
(2)   CALL GKS$SET_WINDOW( HALF, 0.0, 0.9, 0.0, 0.5)
      CALL GKS$SET_VIEWPORT( LOW_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5)

(3)   CALL GKS$SELECT_XFORM( LOW_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the screen.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
```

(continued on next page)

Example 7–2 (Cont.): Controlling Clipping at the World Viewport

```
❹       CALL GKS$SET_CLIPPING( GKS$K_NOCLIP )

C       Draw the same house, using the same windows and viewports, but
C       with clipping disabled. Now you can see the portion of the house
C       that overlaps the window and viewport.
        CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
        CALL GKS$DEACTIVATE_WS( WS_ID )
        CALL GKS$CLOSE_WS( WS_ID )
        CALL GKS$CLOSE_GKS()
        END
```

The following numbers correspond to the numbers in the previous example:

❶ PX contains the house's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the lower right corner of the house (0.4, 0.1).

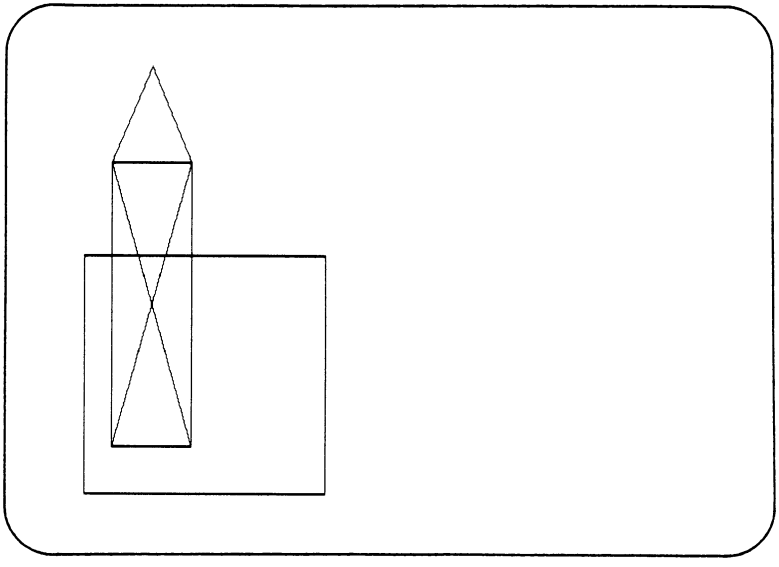
❷ This code sets the normalization window to be half of the default plane and assigns the normalization transformation LOW_LEFT_CORNER (number 1). The tall, thin house is cut in half by the window boundary. The code also sets the viewport to be the lower left corner of the NDC space and assigns this normalization transformation the value 1.

❸ This code selects normalization transformation number 1 which has a smaller window by half and a viewport that is the lower left corner of the NDC space. By default, clipping is enabled; you only see half the house.

❹ Once you disable clipping and redraw the picture, DEC GKS maps the entire house to the NDC space and eventually to the workstation surface.

Figure 7–10 shows the screen of a VT241 terminal after the program has run to completion.

Figure 7–10: Enabling and Disabling Clipping—VT241



ZK-5843-HC

SET VIEWPORT INPUT PRIORITY

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_VIEWPORT_PRIORITY specifies the input priority of a given normalization transformation. If during stroke or locator input DEC GKS encounters a point on overlapping viewports in the NDC space, the viewport input priority list determines which normalization transformation to use when transforming the point back to world coordinate space.

By default, the normalization transformations are ordered in a sequential list so that the transformation number 0 is the highest priority transformation, and the transformation number 255 is the lowest. When you request stroke or locator input, DEC GKS uses the viewport with the highest priority when transforming a point on overlapping viewports.

To change the priority list using GKS$SET_VIEWPORT_PRIORITY, you specify a normalization transformation number whose priority is to be changed, specify an index transformation number, and specify whether the first transformation is the next higher or lower priority.

See Section 7.1.2 for more information concerning input and overlapping viewports.

Syntax

GKS$SET_VIEWPORT_PRIORITY *(transformation_number, reference_trans_number, relative_priority)*

GSVPIP *(xform, ref_xform, rel_prior)*

gsetviewportinputpri *(transform, reference, priority)*

Arguments

transformation_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the normalization transformation number whose priority you want to change.

reference_trans_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This is a normalization transformation number to be used as a reference for changing the priority of the first transformation number. You specify whether or not the first number is of the next higher or lower priority than this reference number. If you specified lower, the first number is placed directly behind this reference number in the sequential priority list. If you specified higher, DEC GKS places the first number directly in front of this reference number in the sequential priority list.

relative_priority

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is a flag that determines whether or not the normalization transformation number is of higher or lower priority than the reference normalization transformation number. The argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_INPUT_PRIORITY_HIGHER	Higher priority
1	GKS$K_INPUT_PRIORITY_LOWER	Lower priority

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-24	DECGKS$_ERROR_NEG_24	Invalid value specified for viewport priority flag in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC or SGOP in routine ****
50	GKS$_ERROR_50	Transformation number is invalid in routine ****

Program Example

Example 7–3 illustrates the use of the function GKS$SET_VIEWPORT_PRIORITY. Following the program example, Figure 7–11 illustrates the program's effect on a VT241 workstation.

Example 7–3: Setting the Input Priority

```
C     This program accepts input twice from the same spot on the
C     workstation surface.  When the input priority is changed,
C     the world coordinates returned are that of the other overlapping
C     viewport.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, ERROR_STATUS, INPUT_MODE, ECHO_FLAG, XFORM,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE, INPUT_STATUS, DEFAULT,
     * LOW_LEFT_CORNER, RIGHT_HALF, DEVICE_NUM, NUM_POINTS,
     * PROMPT_ECHO_TYPE, DATA_RECORD( 1 )
      REAL WORLD_COORD_X, WORLD_COORD_Y,
     * ECHO_AREA( 4 ), PX( 5 ), PY( 5 ),
     * LARGER
      DATA PX / 0.0, 1.0, 1.0, 0.0, 0.0 /
      DATA PY / 0.0, 0.0, 1.0, 1.0, 0.0 /
      DATA DEFAULT / 0 /, DEVICE_NUM / 1 /, LARGER / 0.04 /,
     * RIGHT_HALF / 1 /, LOW_LEFT_CORNER / 1 /, NUM_POINTS / 5 /,
     * WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     When you outline the entire default world coordinate space, you
C     also outline the entire NDC space, the entire workstation window,
C     and the entire workstation viewport.
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )

C     This window and viewport are associated with the
C     normalization transformation number 1.
      CALL GKS$SET_WINDOW( LOW_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
      CALL GKS$SET_VIEWPORT( RIGHT_HALF, 0.5, 1.0, 0.0, 1.0 )

C     Select the new transformation and outline the new windows
C     and viewports.
      CALL GKS$SELECT_XFORM( 1 )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )


C     Assign a value to RECORD_BUFFER_LENGTH: 4 bytes.  On output,
C     this argument should contain the value 0 since
C     GKS$INQ_LOCATOR_STATE does not write anything to the buffer.
      RECORD_BUFFER_LENGTH = 4
❶     CALL GKS$INQ_LOCATOR_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE, ECHO_FLAG,
     * XFORM, WORLD_COORD_X, WORLD_COORD_Y, PROMPT_ECHO_TYPE,
     * ECHO_AREA, DATA_RECORD, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE )
```

(continued on next page)

Example 7–3 (Cont.): Setting the Input Priority

```
❷      PROMPT_ECHO_TYPE = 1
❸      CALL GKS$INIT_LOCATOR( WS_ID, DEVICE_NUM, 0.7, 0.5, DEFAULT,
      * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
      * RECORD_BUFFER_LENGTH )
❹      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
      * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

C      ********************************
C      At this pause, just type RETURN.
C      ********************************
       CALL GKS$REQUEST_LOCATOR( WS_ID, DEVICE_NUM, INPUT_STATUS,
      * XFORM, WORLD_COORD_X, WORLD_COORD_Y )
C      Write the returned world coordinates.
       WRITE(5, *) WORLD_COORD_X, WORLD_COORD_Y
       CALL GKS$SELECT_XFORM( DEFAULT )
       CALL GKS$SET_TEXT_HEIGHT( LARGER )
       CALL GKS$TEXT( 0.01, 0.4, 'Higher priority VP: 0' )

C      Set the current viewport (associated with the selected
C      transformation number 1) to be a higher priority than the
C      default viewport.
❺      CALL GKS$SET_VIEWPORT_PRIORITY( RIGHT_HALF, DEFAULT,
      * GKS$K_INPUT_PRIORITY_HIGHER )

C      Call for input from the same spot.
❻      CALL GKS$INIT_LOCATOR( WS_ID, DEVICE_NUM, 0.7, 0.5, DEFAULT,
      * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
      * RECORD_BUFFER_LENGTH )
       CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
      * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )
```

(continued on next page)

Example 7–3 (Cont.): Setting the Input Priority

```
C     *********************************
C     At this pause, just type RETURN.
C     *********************************
      CALL GKS$REQUEST_LOCATOR( WS_ID, DEVICE_NUM, INPUT_STATUS,
     * XFORM, WORLD_COORD_X, WORLD_COORD_Y )

C     Write the returned world coordinates, this time from the smaller
C     viewport on the right half of the screen.
      WRITE(5, *) WORLD_COORD_X, WORLD_COORD_Y
      CALL GKS$SELECT_XFORM( DEFAULT )
      CALL GKS$TEXT( 0.01, 0.3, 'Higher priority VP: 1')

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The function GKS$INQ_LOCATOR_STATE enables you to change the input data record and other pertinent values that are needed to perform locator input. Locator input returns a single world coordinate point. For more information concerning the types of input in DEC GKS, refer to Chapter 8, Input Functions. For more information concerning the arguments to GKS$INQ_LOCATOR_STATE, refer to Chapter 12, Inquiry Functions.

❷ The prompt and echo types are device independent. Depending on the type of prompt and echo type you choose, you may or may not need to pass an input data record.

In this example, the type of prompt echo is a tracking plus sign. When using this prompt and echo type, pass a dummy data record; DEC GKS does not use the record. For more information, refer to Chapter 8, Input Functions.

❸ The function GKS$INIT_LOCATOR sets the initial values needed to perform locator input, including the position in world coordinates, on which to place the tracking plus sign. This call places the plus sign on the world coordinate (0.7, 0.5). For more information, refer to Chapter 8, Input Functions.

❹ The function GKS$SET_LOCATOR_MODE sets the operating mode of a specific locator class device and allows DEC GKS to accept input. At this point in the program, just press the RETURN key. The program returns the world coordinates from the viewport associated with the transformation number with the higher priority, unity transformation number 0. For more information, refer to Chapter 8, Input Functions.

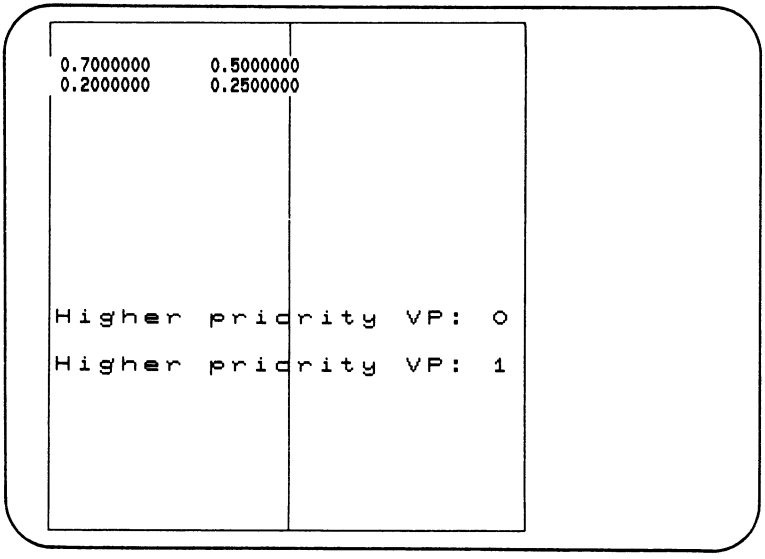
❺ When you last requested input, the normalization transformation number 0 had higher priority. Consequently, the world coordinates returned were from the default window.

To return world coordinates from the window in the lower left corner of the world coordinate plane, whose viewport overlaps the default viewport on the right side of the NDC plane, give the transformation number 1 the higher priority.

❻ When you initialize the tracking plus sign to the exact same spot on the workstation surface, and when you request locator input again, the program returns different world coordinates; it returns the world coordinates from the overlapping viewport associated with the normalization transformation number 1.

Figure 7–11 shows the screen of a VT241 terminal after the program has run to completion.

Figure 7–11: Setting the Input Priority—VT241



ZK-5082-86

SET VIEWPORT

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_VIEWPORT associates viewport boundaries on the NDC plane with a specified normalization transformation number.

By default, DEC GKS uses the range ([0,1] x [0,1]), in NDC coordinates, as the normalization viewport.

DEC GKS maps the normalization window onto the viewport. The image retains its position in the viewport relative to its position in the window, but the image's aspect ratio may differ. Also, if the image extends past the boundary of the window, it may or may not extend past the viewport boundary, depending on whether or not clipping is enabled or disabled. See Section 7.1 for more information concerning normalization transformations and clipping.

Syntax

GKS$SET_VIEWPORT *(transformation_number, minimum_x_value, maximum_x_value, minimum_y_value, maximum_y_value)*

GSVP *(xform, xmin, xmax, ymin, ymax)*

gsetviewport *(transform, viewport)*

Arguments

transformation_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the normalization transformation number. Once you select this number by calling GKS$SELECT_XFORM, any subsequent calls to output functions use the window and viewport associated with this number.

minimum_x_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the lower left corner point of the X value in a rectangular viewport. Make sure that the lower left corner is located within the default normalization viewport range. See Section 7.1 for more information concerning normalization transformations.

maximum_x_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the X value for the upper right corner of the viewport, and with the minimum X and Y values and maximum Y value, determines the rectangular area of the viewport within the NDC space. Make sure that the maximum X and Y values are located within the default normalization viewport boundaries. See Section 7.1 for more information concerning normalization transformations.

minimum_y_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the lower left corner point of the Y value in a rectangular viewport. Make sure that the lower left corner is located within the default normalization viewport range. See Section 7.1 for more information concerning normalization transformations.

maximum_y_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the Y value for the upper right corner of the viewport, and with the minimum X and Y values and maximum X value, determines the rectangular area of the viewport within the NDC space. Make sure that the maximum X and Y values are located within the default normalization viewport boundaries. See Section 7.1 for more information concerning normalization transformations.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC or SGOP in routine ****
50	GKS$_ERROR_50	Transformation number is invalid in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****
52	GKS$_ERROR_52	Viewport is not within the Normalized Device Coordinate unit square in routine ****

Program Example

Refer to Example 7–1 in this section for a program example using a call to GKS$SET_VIEWPORT.

SET WINDOW

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_WINDOW associates normalization window boundaries on the world coordinate plane with a specified normalization transformation number.

By default, DEC GKS uses the range ([0,1] x [0,1]), in world coordinates, as the normalization window.

DEC GKS maps the normalization window onto the viewport. The image retains its position in the viewport relative to its position in the window, but the image's aspect ratio may differ. Also, if the image extends past the boundary of the world window, it may or may not extend past the viewport boundary, depending on whether or not clipping is enabled or disabled. See Section 7.1 for more information concerning normalization transformations and clipping.

Syntax

GKS$SET_WINDOW *(transformation_number, minimum_x_value, maximum_x_value, minimum_y_value, maximum_y_value)*

GSWN *(xform, xmin, xmax, ymin, ymax)*

gsetwindow *(transform, window)*

Arguments

transformation_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the normalization transformation number. Once you select this number by calling GKS$SELECT_XFORM, any subsequent calls to output functions use the window and viewport associated with this number.

minimum_x_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the lower left corner point of the X value in a rectangular window.

maximum_x_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the X value for the upper right corner of the window, and with the minimum X and Y values and maximum Y value, determines the rectangular area of the window within the world coordinate space.

minimum_y_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the lower left corner point of the Y value in a rectangular window.

maximum_y_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the Y value for the upper right corner of the window, and with the minimum X and Y values and maximum X value, determines the rectangular area of the window within the world coordinate space.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC or SGOP in routine ****
50	GKS$_ERROR_50	Transformation number is invalid in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****

Program Example

Refer to Example 7–1 in this section for a program example using a call to GKS$SET_WINDOW.

SET WORKSTATION VIEWPORT

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_WS_VIEWPORT establishes the portion of the workstation surface on which DEC GKS maps the workstation window.

The default workstation viewport is the largest square on the workstation surface, beginning with the lower left corner. If you define a new workstation viewport or window such that the two are not proportionally equivalent, DEC GKS may not use the entire viewport. DEC GKS only uses the portion of the viewport that maintains the shape of the picture in the workstation window. See Section 7.2 for detailed information.

NOTE

If your workstation cannot implement an immediate change to the workstation window or viewport, the surface needs to be regenerated to establish the requested settings. If the surface is regenerated, the surface is cleared and only output primitives stored in segments are redrawn. You lose any primitives not contained in segments. For a detailed discussion of surface regeneration, refer to Chapter 4, Control Functions.

Syntax

GKS$SET_WORKSTATION_VIEWPORT *(workstation_number, minimum_x_value, maximum_x_value, minimum_y_value, maximum_y_value)*

GSWKVP *(workstation_id, xmin, xmax, ymin, ymax)*

gsetwsviewport *(workstation_id, viewport)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is a workstation identifier specified in a previous call to GKS$OPEN_WS.

minimum_x_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the lower left corner point of the X value in a rectangular workstation viewport. Make sure that this point is located within the display surface limits of the specified workstation. You should use the function GKS$INQ_MAX_DS_SIZE, which returns the maximum X and Y values of the workstation display surface. (For more information, refer to Chapter 12, Inquiry Functions.)

maximum_x_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the X value for the upper right corner of the viewport, and with the minimum X and Y values and maximum Y value, determines the rectangular area of the workstation viewport within the device coordinate space. Make sure that the maximum X and Y values are located within the display surface limits of the specified workstation. You should use the function GKS$INQ_MAX_DS_SIZE, which returns the maximum X and Y values of the workstation display surface. (For more information, refer to Chapter 12, Inquiry Functions.)

minimum_y_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the lower left corner point of the Y value in a rectangular workstation viewport. Make sure that this point is located within the display surface limits of the specified workstation. You should use the function GKS$INQ_MAX_DS_SIZE, which returns the maximum X and Y values of the workstation display surface. (For more information, refer to Chapter 12, Inquiry Functions.)

maximum_y_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the Y value for the upper right corner of the viewport, and with the minimum X and Y values and maximum X value, determines the rectangular area of the workstation viewport within the device coordinate space. Make sure that the maximum X and Y values are located within the display surface limits of the specified workstation. You should use the function GKS$INQ_MAX_DS_SIZE, which returns the maximum X and Y values of the workstation display surface. (For more information, refer to Chapter 12, Inquiry Functions.)

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****
54	GKS$_ERROR_54	Workstation viewport is not within the display space in routine ****

Program Example

Example 7–4 illustrates the use of the function GKS$SET_WS_VIEWPORT. Following the program example, Figure 7–12 illustrates the program's effect on a VT241 workstation.

Example 7–4: Establishing a Workstation Viewport

```
C     This program uses the default normalization transformations,
C     generates  the "tall, thin house," updates the screen,
C     changes the workstation viewport to the lower left
C     corner of the VT241 surface, and generates the output again.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, RASTER_Y, ERROR, METERS,
     * RASTER_X, DEFAULT
      REAL PX ( 9 ), PY ( 9 ), PX_2 ( 5 ), PY_2 ( 5 ),
     * DEVICE_MAX_X, DEVICE_MAX_Y
(1)   DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA PX_2 / 0.0, 1.0, 1.0, 0.0, 0.0 /
      DATA PY_2 / 0.0, 0.0, 1.0, 1.0, 0.0 /
      DATA NUM_POINTS / 9 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Outlining the default world window will result in the outlining of
C     the NDC plane, the workstation window, and the workstation
C     viewport, by default.
      CALL GKS$POLYLINE( 5, PX_2, PY_2 )

(2)   CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the screen.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
(3)   CALL GKS$INQ_MAX_DS_SIZE( GKS$K_VT240, ERROR, METERS,
     * DEVICE_MAX_X, DEVICE_MAX_Y, RASTER_X, RASTER_Y )

(4)   CALL GKS$SET_WS_VIEWPORT( WS_ID, 0.0, DEVICE_MAX_X/2.0,
     * 0.0, DEVICE_MAX_Y/2.0 )

C     Update the screen so that the workstation can use the
C     new workstation viewport (as noted in the function
C     description section).
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

C     Outline the default windows and viewports and draw the house again.
      CALL GKS$POLYLINE( 5, PX_2, PY_2 )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
```

(continued on next page)

Example 7–4 (Cont.): Establishing a Workstation Viewport

```
CALL GKS$DEACTIVATE_WS( WS_ID )
CALL GKS$CLOSE_WS( WS_ID )
CALL GKS$CLOSE_GKS()
END
```

The following numbers correspond to the numbers in the previous example:

❶ PX contains the house's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the lower right corner of the house (0.4, 0.1).

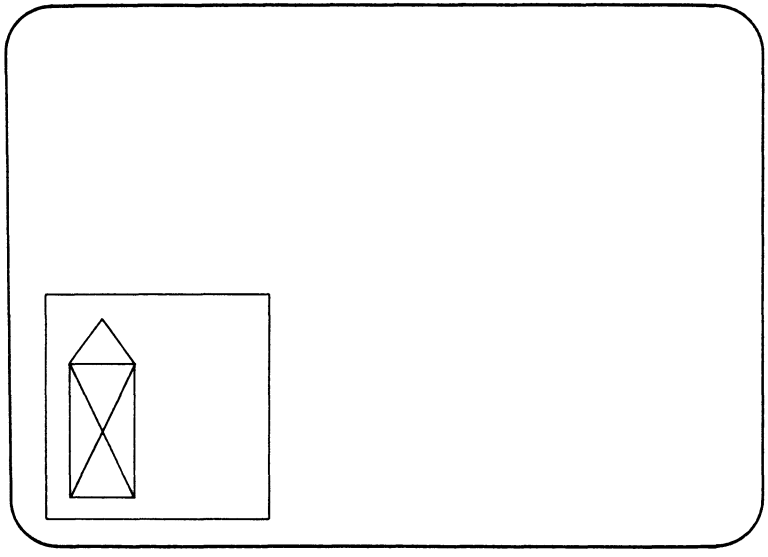
❷ This code assumes the default normalization transformation. DEC GKS maps the default window to the default viewport, and then maps the default workstation window to the default viewport.

❸ The function GKS$INQ_MAX_DS_SIZE returns the maximum X and Y display surface coordinates in the arguments DEVICE_MAX_X and DEVICE_MAX_Y. For more information concerning the other arguments, refer to the GKS$INQ_MAX_DS_SIZE function description in Chapter 12, Inquiry Functions.

❹ The function GKS$SET_WS_VIEWPORT changes the workstation viewport to the lower left corner of the screen. The picture being displayed is still the same (the default workstation window), but the space on the workstation surface used to display the same picture has changed.

Figure 7–12 shows the screen of a VT241 terminal after the program has run to completion.

Figure 7–12: Establishing a Workstation Viewport—VT241



ZK-5083-86

SET WORKSTATION WINDOW

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_WS_WINDOW establishes the portion of the composed picture (on the NDC plane) that DEC GKS maps to the current workstation viewport. Despite the current value of the clipping flag, DEC GKS clips all pictures at the workstation window boundary.

By default, DEC GKS uses the entire picture, mapping the default workstation window range ([0,1] x [0,1]), onto the largest square that the workstation can produce.

NOTE

If your workstation cannot implement an immediate change to the workstation window or viewport, the surface needs to be regenerated to establish the current settings. If the surface is regenerated, the surface is cleared and only output primitives stored in segments are redrawn. You lose any primitives not contained in segments. For a detailed discussion of surface regeneration, refer to Chapter 4, Control Functions.

Syntax

GKS$SET_WS_WINDOW *(workstation_id, minimum_x_value, maximum_x_value, minimum_y_value, maximum_y_value)*

GSWKWN *(workstation_id, xmin, xmax, ymin, ymax)*

gsetwswindow *(workstation_id, window)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is a workstation identifier specified in a previous call to GKS$OPEN_WS.

minimum_x_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the lower left corner point of the X value in a rectangular workstation window, in NDC points. Make sure that this point is located within the default workstation window boundary. See Section 7.2 for more information concerning workstation transformations.

maximum_x_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument specifies the lower left corner point of the Y value in a rectangular workstation window, in NDC points. Make sure that this point is located within the default workstation window boundary. See Section 7.2 for more information concerning workstation transformations.

minimum_x_value

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument specifies the upper right corner of the X value of the window, and used with the maximum X and Y values, and the minimum Y value, determines the rectangular area of the workstation window within the NDC space. Make sure that the maximum X and Y values are located within the

default workstation window boundary. See Section 7.2 for more information concerning workstation transformations.

minimum_y_value

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument specifies the upper right corner of the Y value of the window, and used with the maximum X and Y values, and the minimum X value, determines the rectangular area of the workstation window within the NDC space. Make sure that the maximum X and Y values are located within the default workstation window boundary. See Section 7.2 for more information concerning workstation transformations.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****

Error Number	Completion Status Code	Message
36	GKS$_ERROR_36	Specified workstation is Workstation Independent Segment Storage in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****
53	GKS$_ERROR_53	Workstation window is not within the Normalized Device Coordinate unit square in routine ****

Program Example

Example 7–5 illustrates the use of the function GKS$SET_WS_WINDOW. Following the program example, Figure 7–13 illustrates the program's effect on a VT241 workstation.

Example 7–5: Establishing a Workstation Window

```
C     This program uses the default normalization transformations,
C     generates  the "tall, thin house," updates the screen,
C     changes the workstation window to the lower half of the
C     default workstation window, and generates the output again.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, NUM_POINTS, DEFAULT
❶     REAL PX ( 9 ), PY ( 9 ), PX_2 ( 5 ), PY_2 ( 5 )
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA PX_2 / 0.0, 1.0, 1.0, 0.0, 0.0 /
      DATA PY_2 / 0.0, 0.0, 1.0, 1.0, 0.0 /
      DATA NUM_POINTS / 9 /, DEFAULT / 0 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
```

(continued on next page)

Example 7–5 (Cont.): Establishing a Workstation Window

```
C     Outlining the default world window will result in the outlining of
C     the NDC plane, the workstation window, and the workstation
C     viewport, by default.
      CALL GKS$POLYLINE( 5, PX_2, PY_2 )

❷     CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the screen.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
C     Set a new workstation window that is the lower half of the
C     default workstation window.
❸     CALL GKS$SET_WS_WINDOW( WS_ID, 0.0, 1.0, 0.0, 0.5 )

C     UPDATE the surface so that DEC GKS can use the new
C     workstation window (as noted in the function description
C     section).
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

C     Outline the default NDC space and draw the house again.
      CALL GKS$POLYLINE( 5, PX_2, PY_2 )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

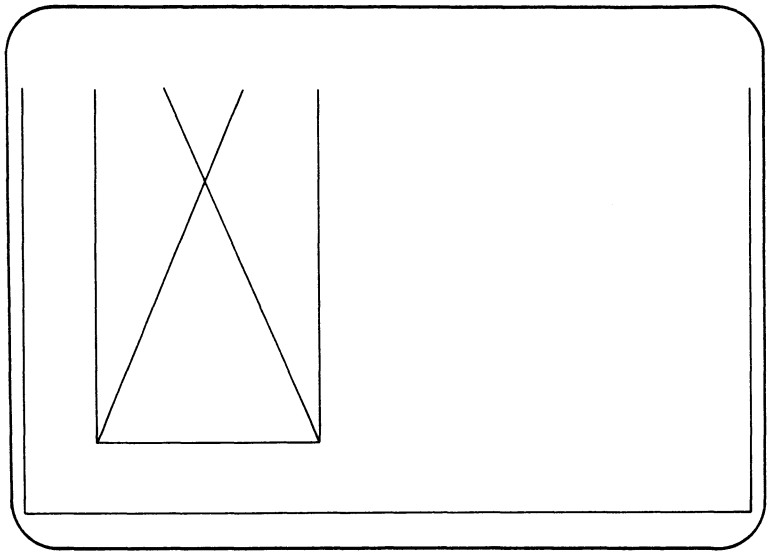
❶ PX contains the house's X world coordinate values and PY contains the Y world coordinate values. For example, the first element in both arrays specifies the lower right corner of the house (0.4, 0.1).

❷ This code assumes the default normalization transformation. DEC GKS maps the default window to the default viewport and then maps the default workstation window to the default workstation viewport.

❸ The function GKS$SET_WS_WINDOW changes the workstation window to be the lower half of the NDC space. The picture is now half of what it was and will now be mapped onto the largest rectangle on the display surface, starting at lower left point within the current workstation viewport, that maintains the aspect ratio. Notice how DEC GKS does not map the current workstation window onto the entire workstation viewport, so as to maintain the shape of the picture.

Figure 7–13 shows the screen of a VT241 terminal after the program has run to completion.

Figure 7–13: Establishing a Workstation Window—VT241



Chapter 8

Input Functions

The DEC GKS input functions allow an application program to accept input from a user. The following list presents the input functions by category:

Category	GKS Functions
Initialization	GKS$INIT_CHOICE, GKS$INIT_LOCATOR, GKS$INIT_PICK, GKS$INIT_STRING, GKS$INIT_STROKE, GKS$INIT_VALUATOR
Mode Control	GKSSET_CHOICE_MODE, GKSSET_LOCATOR_MODE, GKSSET_PICK_MODE, GKSSET_STRING_MODE, GKSSET_STROKE_MODE, GKSSET_VALUATOR_MODE
Request Mode	GKS$REQUEST_CHOICE, GKS$REQUEST_LOCATOR, GKS$REQUEST_PICK, GKS$REQUEST_STRING, GKS$REQUEST_STROKE, GKS$REQUEST_VALUATOR
Sample Mode	GKS$SAMPLE_CHOICE, GKS$SAMPLE_LOCATOR, GKS$SAMPLE_PICK, GKS$SAMPLE_STRING, GKS$SAMPLE_STROKE, GKS$SAMPLE_VALUATOR
Event Mode	GKS$AWAIT_EVENT, GKS$FLUSH_DEVICE_EVENTS, GKSGET_CHOICE, GKSGET_LOCATOR, GKSGET_PICK, GKSGET_STRING, GKSGET_STROKE, GKSGET_VALUATOR

8.1 Physical and Logical Input Devices

There are many physical devices that can provide input to a program. Common physical devices include a terminal keyboard, a tablet, and a mouse. You can translate the information obtained from these physical devices in several ways. For instance, a letter typed at the keyboard can represent a decimal ASCII value, a character value, a character string containing one letter, or many other values depending on how the program stores and then interprets the input.

Similarly, DEC GKS accepts several types of input. DEC GKS accepts real numbers (as real values, coordinate points, or as a series of coordinate points), integers (as choice numbers, segment names, or pick identifiers), and character strings. To translate and store input from the physical devices according to the DEC GKS data types, DEC GKS defines *logical input devices*. Logical input devices are abstractions of the physical input devices that allow the user to input data, using a similar visual interface, despite the differences in physical devices. The value returned from a logical input device is called the device's *measure*.

DEC GKS defines the logical input devices by the following components:

- A workstation identifier
- An input class
- A logical device number

The following sections discuss each component in detail.

8.1.1 The Workstation Identifier

The workstation identifier specifies an open workstation, of category GKS$K_WSCAT_INPUT or GKS$K_WSCAT_OUTIN, on which one or more physical input devices are present. For instance, a single workstation supporting input may use both a mouse and a keyboard for input.

The capabilities of a specified workstation determine the prompt and echo types available, and the methods used for signaling the acceptance of an input value (called *input triggers*). A cursor is an example of a prompt (see Section 8.2 for a detailed discussion of prompts and echo types). Pressing the RETURN key to signal the acceptance of a value is an example of a trigger.

8.1.2 The Input Class

The *input class* is the second input component, and it tells DEC GKS the data type of the information to be entered. The six input classes are as follows:

- Locator
- Stroke
- Valuator
- Choice
- String
- Pick

A locator class device first positions a prompt on the workstation surface. The user can then move the prompt (on a VT241, the user can press the arrow keys), and if you are using an applicable input mode, trigger the input device (on a VT241, by pressing the RETURN key). This input class returns two real numbers that represent world coordinate values. DEC GKS transforms the input point from a device coordinate point to a normalized device coordinate (NDC) point, and then from an NDC point to a corresponding world coordinate point.

A stroke class device also positions a prompt on the workstation surface. The user can then move the prompt (on a VT241, by pressing the arrow keys) and, when choosing a desired device coordinate point, the user *signals* DEC GKS (on a VT241, by pressing the space bar; on the VAXstations, you specify a distance vector and the user enters points by pressing a button on the mouse). The user keeps choosing points until the desired sequence is entered. This input class returns a sequence of real numbers that are the corresponding world coordinate values of the stroke. DEC GKS transforms the input points from device coordinate points to NDC points, and then from NDC points to corresponding world coordinate points.

For more information concerning the DEC GKS coordinate systems, refer to Chapter 7, Transformation Functions.

A valuator class device creates a picture on the workstation surface that represents a real number or a series of real number values. You specify the lowest and highest values. For several workstations, the valuator class may look like a radio dial with a pointer to a current value. The user moves the cursor up and down the scale (on a VT241, using the arrow keys) until it is positioned as desired (on a VT241, pressing the RETURN key can trigger this class of device). This input class returns the real number representing the pointer's last position on the scale.

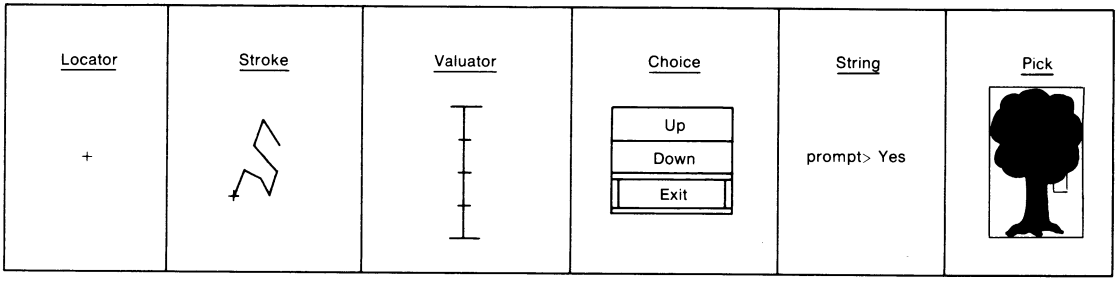
A choice class device creates a picture on the workstation surface that lists a series of choices. The user sees the choices as you label them; the choices are represented internally by integer values. For several workstations, the choices can look like a menu, with the currently selected choice highlighted (on a VT241, in complement mode). The user moves among the choices (on a VT241, by pressing the arrow keys) until the user's choice is highlighted. When triggered (on a VT241, by pressing the RETURN key), this input class returns the integer representing the selected choice.

A string class device creates an area on the workstation surface on which the user can enter a character string. Optionally, you can provide an initial string to prompt the user. DEC GKS appends the input string to the initial string. The user can enter a string as large as the defined input buffer. On many workstations, pressing the RETURN key can trigger the string input devices.

A pick class device positions a prompt on the workstation surface. The user moves the prompt among the segments on the workstation surface (on a VT241, by using the arrow keys), and if you are using an applicable input mode, the user can trigger the device (on a VT241, by pressing the RETURN key). The pick device returns integers that represent the name of the picked segment or the *pick identifier* associated with portions of a segment. (For detailed information concerning pick identifiers, refer to Chapter 9, Segment Functions.)

Figure 8–1 illustrates possible visual interfaces for the logical input classes.

Figure 8–1: Logical Input Classes



Differences between workstations' implementations of logical input classes may be significant. For example, using a stroke input device on a VAXstation, you can specify X and Y device coordinate change vectors to tell the graphics handler when to add another device coordinate point to the stroke. When the user specifies a point whose distance from the last entered point exceeds both the specified X and Y vectors, the input device accepts the point as the next point in the stroke. This affects the smoothness of the line, allowing you to create relatively curved shapes instead of jagged lines. If you specify a relatively short X and Y difference, DEC GKS accepts many of the input points as you move the mouse.

In contrast, on the VT241, you must move the arrow keys and signal each time you have reached a point you want to be a part of the stroke. If the point exceeds the X and Y change vector values, then the graphics handler accepts the points.

8.1.3 The Device Number

DEC GKS also uses the *logical device number* to differentiate between various methods of entering the same class of data on the same workstation. For instance, a workstation may use both the mouse and the arrow keys on a keyboard as two distinct stroke logical input devices. You can distinguish between the two input devices by their logical device number. For instance, the graphics handler could assign the logical device number 1 to the stroke device using the mouse, and could assign the number 2 to the stroke device using the arrow keys on the keyboard. When you request input on such a workstation, you specify whether you wish to use stroke logical input device 1 or 2 (the mouse or the keyboard). You need to tell the user which physical devices (such as keyboard keys, or mouse buttons) control the particular logical input device.

8.2 Prompt and Echo Types

There are differences in the way a single workstation can prompt the user and can echo the input when using the same logical device. Some differences may be subtle. For example, a workstation may use either a plus sign or a set of cross hairs as a prompt for a single locator device, both triggered by pressing the RETURN key. To distinguish between different visual interfaces available for a single logical input device, that device can have a number of *prompt and echo types.*

For example, the VT241 graphics handler accepts seven different prompt and echo types for its locator input devices. The prompt can be any of the following:

- A box (similar in appearance to the pick aperture used for pick input)
- A tracking plus sign (+)
- A cross hair
- A tracking cross (X)
- A line from the initial locator position to the current locator position
- A rectangle whose diagonal connects the initial and current positions
- A numeric representation of the current locator position

Since the graphics handlers use DEC GKS primitives such as lines, markers, and fill areas to construct input prompts, the graphics handler optionally uses additional information that determines how the prompt and echoed input appear on the surface. For instance, a handler may use a polyline output attribute that would affect the appearance of cross hairs on the surface. The

requirements depend on the needs of the different prompt and echo types on different physical devices.

To pass information to meet the requirements of a certain prompt and echo type on a given logical input device, you use the *input data record.* You can either use the default data record or specify a new data record to one of the input initializing functions (GKS$INIT_LOCATOR, GKS$INIT_STROKE, and so forth).

See Section 8.2.1 for a detailed discussion of input data records. To review the available prompt and echo types for a given logical input device on your workstation, use the appropriate workstation description table inquiry function or refer to Appendix J, DEC GKS Specific Input Values.

8.2.1 Input Data Records

Due to the needs of a given prompt and echo type for a logical input class, you need to pass an input data record if you choose to call one of the initializing input functions (GKS$INIT_LOCATOR, GKS$INIT_STROKE, and so forth). The data record contains information relevant to the input prompt interface. For instance, the input data record for a locator logical input device may specify output attributes that affect the thickness or color of cross hairs on the workstation surface.

When the GKS standard describes the input data records, it specifies required and nonrequired components of the data record. If a component is required, you can be certain that all GKS graphics handlers use that component of the data record. If an input data record component is nonrequired, the device handler must be able to *accept* that component, but may or may not *use* that component when generating the input prompt and echo. For instance, if polyline color is a nonrequired part of the data record, the GKS implementation cannot generate an error if it encounters the component, but does not have to change the color of the prompt on the workstation surface.

NOTE

In almost all cases, the DEC GKS supported graphics handlers require that you pass the complete GKS standard data record. Depending on the device, the graphics handler may or may not use all of the components. For variable-length data records, you must be sure to pass the correct size of the data record. For detailed information concerning the required data record size, the initial data record values, or the components actually used by a particular graphics handler, refer to Appendix J, DEC GKS Specific Input Values.

The following sections describe the input data records defined by the GKS standard. Input data records are defined according to the logical input class and the prompt and echo type. The tables list the GKS standard data records. The column labelled Required either contains an R for required or an N for nonrequired. If you have a general understanding of input data records, you may only need to skim these sections.

Choice Class

The GKS standard defines the following prompt and echo type values:

Echo Type Number	Description
1	Designate the current choice integer value using an implementation-specific technique. The device can use any portion of the data record (for information specific to your device, refer to the *DEC GKS Device Specifics Reference Manual.*
2	Use the device's capability for prompting. GKS compares the number of prompts requested with the number of maximum prompts on the device. The data record specifies whether each prompt is turned on or turned off.
3	Using an appropriate technique, allow the user to select one of a group of choices, each choice labeled with a string.
4	Using an alphanumeric keyboard, allow the user to select one of a group of choices, each choice labeled with a string.
5	Map a segment to the echo area and map the segment's pick identifiers to a numeric choice value.

Prompt and echo type numbers greater than or equal to the number 6 are reserved for future standardization. Numbers less than or equal to 0 are device dependent.

Choice Class: Prompt and Echo Type 1

The GKS standard does not define a data record for this prompt and echo type. The format of the record is device dependent.

Choice Class: Prompt and Echo Type 2

Position	Data Type	Required	Description
1	Integer	R	Number of choice alternatives.
2	Array (integer)	R	Array of prompts turned either on or off. GKS$K_CHOICE_PROMPT_ON (0) or GKS$K_CHOICE_PROMPT_OFF (1).

Choice Class: Prompt and Echo Types 3 and 4

Position	Data Type	Required	Description
1	Integer	R	Number of choice strings.
2	Array (string)	R	Array of strings as choice labels.

NOTE

DEC GKS supported graphics handlers implement the data record for choice prompt and echo type 3 using three components instead of two. For detailed information, refer to Appendix J, DEC GKS Specific Input Values.

Choice Class: Prompt and Echo Type 5

Position	Data Type	Required	Description
1	Integer	R	Segment name.
2	Integer	R	Number of choice alternatives.
3	Array (integer)	R	Array of pick identifiers.

Locator Class

The GKS standard defines the following prompt and echo type values:

Echo Type Number	Description
1	Mark the current location in an implementation-specific manner. The device can use any portion of the data record (for information specific to your device, refer to the *DEC GKS Device Specifics Reference Manual.*
2	Mark the current location by using a vertical and horizontal line as crosshairs.
3	Mark the current location using a tracking cross.
4	Mark the current location using a line connecting the current location to the initial location.
5	Mark the current location using a rectangle whose diagonal is the current location and the initial location.
6	Mark the current location by displaying a digital representation of the location.

Prompt and echo type numbers greater than or equal to the number 7 are reserved for future standardization. Numbers less than or equal to 0 are device dependent.

Locator Class: Prompt and Echo Types 1, 2, 3, 6

The GKS standard does not define a data record. You pass a null data record to the initializing input function.

Locator Class: Prompt and Echo Type 4

Position	Data Type	Required	Description
1	Integer	N	Attribute control flag. GKS$K_ACF_CURRENT (0) or GKS$K_ACF_SPECIFIED (1). Use the currently set output attributes or specify new attributes in this data record.

If component 1 is GKS$K_ACF_SPECIFIED:

Position	Data Type	Required	Description
2	Integer	N	Line type aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
3	Integer	N	Line width scale factor aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
4	Integer	N	Polyline color index aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
5	Integer	N	Polyline index.
6	Integer	N	Line type index.
7	Real	N	Line width scale factor.
5	Integer	N	Polyline color index.

Locator Class: Prompt and Echo Type 5

Position	Data Type	Required	Description
1	Integer	N	Polyline/fill area control flag. GKS$K_ACF_POLYLINE (0) or GKS$K_ACF_FILL_AREA (1). Use a polyline or a filled area to draw the rectangle whose diagonal connects the current and initial points.
2	Integer	N	Attribute control flag. GKS$K_ACF_CURRENT (0) or GKS$K_ACF_SPECIFIED (1). Use the currently set output attributes or specify new attributes in this data record.

If component 1 is GKS$K_ACF_POLYLINE and component 2 is GKS$K_ACF_SPECIFIED:

Position	Data Type	Required	Description
3	Integer	N	Line type aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
4	Integer	N	Line width scale factor aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
5	Integer	N	Polyline color index aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
6	Integer	N	Polyline index.
7	Integer	N	Line type index.
8	Real	N	Line width scale factor.
9	Integer	N	Polyline color index.

If component 1 is GKS$K_ACF_FILL_AREA and component 2 is GKS$K_ACF_SPECIFIED:

Position	Data Type	Required	Description
3	Integer	N	Fill area interior style aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
4	Integer	N	Fill area style index aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
5	Integer	N	Fill area color index aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
6	Integer	N	Fill area index.
7	Integer	N	Fill area interior style. GKS$K_INTSTYLE_HOLLOW (0), GKS$K_INTSTYLE_SOLID (1), GKS$K_INTSTYLE_PATTERN (2), or GKS$K_INTSTYLE_HATCH (3)
8	Integer	N	Fill area style index.
9	Integer	N	Fill area color index.

Pick Class

The GKS standard defines the following prompt and echo type values:

Echo Type Number	Description
1	Use an implementation-defined technique to highlight at least the picked primitive. The device can use any portion of any of the data record (for information specific to your device, refer to the *DEC GKS Device Specifics Reference Manual.*
2	Echo the group of primitives that share the same pick identifier as the picked primitive.
3	Echo the whole segment containing the picked primitive.

Prompt and echo type numbers greater than or equal to the number 4 are reserved for future standardization. Numbers less than or equal to 0 are device dependent.

Pick Class: All Prompt and Echo Types

The GKS standard does not define a data record for this logical input device. The pick data record is device dependent.

String Class

The GKS standard defines the following prompt and echo type values:

Echo Type Number	Description
1	Display the current string value in the echo area.

Prompt and echo type numbers greater than or equal to the number 2 are reserved for future standardization. Numbers less than or equal to 0 are device dependent.

String Class: Prompt and Echo Type 1

Position	Data Type	Required	Description
1	Integer	R	Input buffer size in number of characters.
2	Integer	R	Initial cursor position within the string. The initial position must follow the formula: 1 <= *initial_position* <= *length_initial_string*

Stroke Class

The GKS standard defines the following prompt and echo type values:

Echo Type Number	Description
1	Display the stroke using implementation-defined techniques. The device can use any portion of the data record (for information specific to your device, refer to the *DEC GKS Device Specifics Reference Manual.*
2	Display a digital representation of the current stroke position 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.

Prompt and echo type numbers greater than or equal to the number 5 are reserved for future standardization. Numbers less than or equal to 0 are device dependent.

Stroke Class: Prompt and Echo Type 1 and 2

Position	Data Type	Required	Description
1	Integer	R	Input buffer size in number of points.
2	Integer	N	Editing position expressed as a stroke point.
3	Real	N	X world coordinate change vector.
4	Real	N	Y world coordinate change vector.
5	Real	N	Time interval, in seconds.

Stroke Class: Prompt and Echo Type 3

Position	Data Type	Required	Description
1	Integer	R	Input buffer size, in number of stroke points.
2	Integer	N	Editing position expressed as a stroke point.
3	Real	N	X world coordinate change vector.
4	Real	N	Y world coordinate change vector.
5	Real	N	Time interval, in seconds.
6	Integer	N	Attribute control flag. GKS$K_ACF_CURRENT (0) or GKS$K_ACF_SPECIFIED (1). Use the currently set output attributes or specify new attributes in this data record.

If component 6 is GKS$K_ACF_SPECIFIED:

Position	Data Type	Required	Description
7	Integer	N	Marker type aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
8	Integer	N	Marker size scale factor aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
9	Integer	N	Polymarker color index aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
10	Integer	N	Polymarker index.
11	Integer	N	Marker type index.
12	Real	N	Marker size scale factor.
13	Integer	N	Polymarker color index.

Stroke Class: Prompt and Echo Type 4

Position	Data Type	Required	Description
1	Integer	R	Input buffer size, in number of stroke points.
2	Integer	N	Editing position expressed as a stroke point.
3	Real	N	X world coordinate change vector.
4	Real	N	Y world coordinate change vector.
5	Real	N	Time interval, in seconds.
6	Integer	N	Attribute control flag. GKS$K_ACF_CURRENT (0) or GKS$K_ACF_SPECIFIED (1). Use the currently set output attributes or specify new attributes in this data record.

If component 6 is GKS$K_ACF_SPECIFIED:

Position	Data Type	Required	Description
7	Integer	N	Line type aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
8	Integer	N	Line width scale factor aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
9	Integer	N	Polyline color index aspect source flag. GKS$K_ASF_BUNDLED (0) or GKS$K_ASF_INDIVIDUAL (1).
10	Integer	N	Polyline index.
11	Integer	N	Line type index.
12	Real	N	Line width scale factor.
13	Integer	N	Polyline color index.

Valuator Class

The GKS standard defines the following prompt and echo type values:

Echo Type Number	Description
1	Designate the current value using implementation-specific techniques. The device can use any portion of the data record (for information specific to your device, refer to the *DEC GKS Device Specifics Reference Manual.*
2	Display a graphical representation of the current value (such as a dial or pointer).
3	Display a digital representation of the current value.

Prompt and echo type numbers greater than or equal to the number 4 are reserved for future standardization. Numbers less than or equal to 0 are device dependent.

Valuator Class: All Prompt and Echo Types

Position	Data Type	Required	Description
1	Real	R	Low value of the numeric range.
2	Real	R	High value of the numeric range.

8.2.1.1 Using an Input Data Record

Of the input data records described in the previous section, the DEC GKS supported graphics handlers must use all required components, but can use any number of the nonrequired components. Appendix J, DEC GKS Specific Input Values, describes the data records required by the DEC GKS supported handlers for each of the supported prompt and echo types, for each of the logical input devices.

As shown in the previous section, the input data record components can be of several different data types. Most languages provide a way for you to declare a contiguous record that can contain items of different data types, but when using some languages, you need to use language-specific means to create such a construct.

For tutorial information concerning the use of input data records, refer to the *DEC GKS Reference Manual*. For information concerning data record declarations, refer to the code examples in this chapter.

8.3 Input Inquiries

When using the DEC GKS input functions, you may need to inquire from the workstation description table, workstation state list, or both. If you require default values, inquire from the description table. If you require the currently set values, inquire from the state list.

The following sections describe inquiry function programming technique.

8.3.1 Default and Current Input Values

If you do not want your application to set all of the input values individually before calling one of the initializing functions (GKS$INIT_LOCATOR, GKS$INIT_STROKE, and so forth), you can pass the input variables to one of two inquiry functions. To obtain default input values, you call the functions GKS$INQ_DEF_LOCATOR_DATA, GKS$INQ_DEF_STROKE_DATA, and so forth. To obtain the current input values, you call the functions GKS$INQ_LOCATOR_STATE, GKS$INQ_STROKE_STATE, and so forth.

When calling those functions, you need to use caution when passing the data record buffer size to the inquiry functions. The buffer size is a modifiable variable (read/write), and when passed to the inquiry function, it must contain the size of the buffer in order for the inquiry function to properly return the contents of the data record.

After the function call, DEC GKS writes the amount of the buffer actually used. You can compare this value to the data record size to see if DEC GKS had to truncate the record when writing it to the buffer. If DEC GKS truncated the data record, you need to decide whether to continue execution or to alter the buffer size so that the entire record fits.

For tutorial information concerning the use of input data records and their buffers, refer to the *DEC GKS Reference Manual*. For information concerning the lengths of data record buffers, refer to the code examples in this chapter.

8.3.2 Device-Independent Programming

Depending on the type of input you use, you may need to call a large number of inquiry functions to create a device-independent program. For instance, you need to check the following information:

- The level of GKS, which determines the supported input operating modes. This information is important for applications that need to be transported to other systems. (DEC GKS is a level 2c implementation.)
- The category of the workstation.
- The number of input devices of a given class supported by the workstation.
- The prompt and echo types supported by a given workstation.
- The maximum possible echo area available on a given workstation.
- The data record information for a given workstation using a specified prompt and echo type (see Section 8.3.1 for detailed information).

For information concerning device-independent programming technique, refer to the *DEC GKS User Manual*. All of the code examples in this book are device independent.

The following list presents the inquiry functions that you can use to obtain input information when writing device-independent code.

GKS$INQ_CHOICE_STATE
GKS$INQ_CURRENT_XFORMNO
GKS$INQ_DEF_CHOICE_DATA
GKS$INQ_DEF_LOCATOR_DATA
GKS$INQ_DEF_PICK_DATA
GKS$INQ_DEF_STRING_DATA
GKS$INQ_DEF_STROKE_DATA
GKS$INQ_LOCATOR_STATE
GKS$INQ_MAX_DS_SIZE
GKS$INQ_OPEN_WS
GKS$INQ_PICK_STATE
GKS$INQ_STRING_STATE
GKS$INQ_STROKE_STATE
GKS$INQ_VALUATOR_STATE

GKS$INQ_DEF_VALUATOR_DATA
GKS$INQ_INPUT_DEV
GKS$INQ_INPUT_QUEUE_OVERFLOW
GKS$INQ_LEVEL
GKS$INQ_WS_CATEGORY
GKS$INQ_WS_XFORM
GKS$INQ_XFORM

8.4 Overlapping Viewports

This section assumes that you have a knowledge of the DEC GKS coordinate systems. You may want to review Chapter 7, Transformation Functions, before reading further.

When defining normalization viewports, it is possible to cause them to overlap on the NDC plane. You must consider the effects this has during input requests. To illustrate the need for a viewport priority list for use during input, consider two normalization viewports: the default viewport ([0,1] x [0,1]) of the unity transformation, and a viewport, belonging to normalization transformation number 1, having the range ([0.5, 1] x [0.5, 1]) in NDC values. The viewport of normalization transformation number 1 overlaps the right half of the default viewport.

During stroke and locator input, the user positions the cursor on the device surface and returns one point or a series of points in device coordinates. DEC GKS translates the device coordinates to NDC points, and then uses the viewport input priority to determine which normalization transformation to use when translating the points to world coordinates.

To decide which normalization viewport has a higher input priority, DEC GKS maintains a priority list. By default, DEC GKS assigns the highest priority to the unity transformation (0). The viewports of all remaining transformations decrease in priority as their transformation numbers increase (viewport 0 higher than viewport 1, 1 higher than 2, 2 higher than 3, and so forth).

When using a locator class device, DEC GKS uses the normalization transformation of the highest input priority that contains the input point. When using stroke input, DEC GKS uses the normalization transformation of the highest priority that contains all of the points in the stroke. Since a locator or stroke input device could not return device coordinate points that could fall outside of the default normalization viewport ([0,1] x [0,1]), the unity transformation can always be used to transform stroke input data.

For more information concerning transformations and viewport priority, refer to Chapter 7, Transformation Functions.

8.5 Input Operating Modes

Using DEC GKS, you can use any of the logical input devices in any of the following three input operating modes:

- Request mode
- Sample mode
- Event mode

If you need to have the application program work synchronously with the input process (if the application must pause to wait for input to be complete), then you can use request mode. Request mode is the only input mode that you can use without first calling one of the GKS$SET_class_MODE functions.

If you need to have the application program work asynchronously with the input process (if the application must run while the user enters input), then you can use sample or event mode. Sample and event mode differ. Using sample mode, the application takes the current measure of an input device without the user having to trigger. While in event mode, the device handler places triggered input values in a time-ordered queue to be accessed when the application chooses.

To change the input operating mode for a given device you call one of the functions GKS$SET_LOCATOR_MODE, GKS$SET_STROKE_MODE, and so forth. These functions serve the second purpose of enabling and disabling echoing of the input prompt. This feature is useful when the DEC GKS echo types are inadequate and you need to echo the input in an application-specific manner.

By default, all device prompts are active at once. For instance, if you press the arrow keys, you alter all prompts on the workstation surface whose devices use the arrow keys. Device handlers can provide methods for the user to deactivate all prompts except one, for each logical input device. In this way, the user can *cycle* through the devices, changing only one measure at a time, in some device-specific order. For more information on cycling logical input devices, refer to Appendix J, DEC GKS Specific Input Values.

NOTE

You cannot cycle past a device whose echoing is disabled. (Normally, notification of the device's turn in the cycle is the displaying of the device's prompt.) Using the corresponding physical device will always alter the measure of a nonechoing device. For instance, if you use pick device 1 on the VT241 while disabling its echoing, pressing the arrow keys always changes the measure of this

device no matter how you cycle through the remaining prompting devices.

The following sections describe each of the input operating modes in greater detail.

8.5.1 Request Mode

In request mode, the application program pauses, and DEC GKS waits for the user either to trigger the end of input or to perform a break. You can use a logical input device in request mode without calling GKS$SET_LOCATOR_MODE, GKS$SET_STROKE_MODE, and so forth, as long as you have not previously set the device to some other mode (request mode is the DEC GKS default mode).

In order to initialize a logical input device, you must make sure that the device's prompt does not currently appear on the workstation surface. To initialize a device, the device must be in request mode.

You can place any or all of the supported logical input devices in request mode at any one time, but you can only request input from one device at a time. You request input, from a specified logical input device on a specified workstation, by calling one of the functions GKS$REQUEST_CHOICE, GKS$REQUEST_LOCATOR, GKS$REQUEST_PICK, GKS$REQUEST_STRING, GKS$REQUEST_STROKE, or GKS$REQUEST_VALUATOR. Once you request input by calling one of the GKS$REQUEST_class functions, the input prompt appears on the workstation surface (if echoing is enabled).

In request mode, there are several ways to trigger or break a request for input. If the user triggers the logical input device (as described in Appendix J, DEC GKS Specific Input Values), DEC GKS writes the value GKS$K_STATUS_OK to the request function input status argument. If the user performs a break during the request for input, which may be a different action on different workstations, DEC GKS writes the value GKS$K_STATUS_NONE to the request function input status argument.

Choice and pick logical input devices allow the user an option other than returning data or breaking input. These logical devices allow the user to end the input process without choosing or picking. If the user triggers the input device without moving the prompt, DEC GKS returns one of the appropriate values, GKS$K_STATUS_NOCHOICE or GKS$K_STATUS_NOPICK, to the input status argument. (DEC GKS also returns GKS$K_STATUS_NOPICK if the user is not currently positioning the aperture on a segment.)

Example 8–1 illustrates the use of the locator logical input device in request mode. Following the program example, Figure 8–2 illustrates the program's effect on a VT241 workstation.

Example 8–1: Using a Locator Logical Input Device in Request Mode

```
C     This program initializes and requests locator input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
❶     INTEGER WS_ID, DATA_RECORD( 1 ), PROMPT_ECHO_TYPE,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, XFORM,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE, INPUT_STATUS,
❷    * DEVICE_NUM
      REAL ECHO_AREA( 4 ), WORLD_X, WORLD_Y
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
❸     CALL GKS$INQ_LOCATOR_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE,
     * ECHO_FLAG, XFORM, WORLD_X, WORLD_Y,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

❹     PROMPT_ECHO_TYPE = 1

C     Since the device is in request mode by default, initialize the device.
❺     CALL GKS$INIT_LOCATOR( WS_ID, DEVICE_NUM, WORLD_X,
     * WORLD_Y, XFORM, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH )

❻     CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

❼     CALL GKS$REQUEST_LOCATOR( WS_ID, DEVICE_NUM, INPUT_STATUS,
     * XFORM, WORLD_X, WORLD_Y )

C     Output the input locator position, in world coordinates.
      WRITE(6,*) WORLD_X, WORLD_Y

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example.

❶ The DEC GKS VT241 handler ignores the data record for all supported locator prompt and echo types. This is a dummy argument.

❷ The echo area variable is an array of real numbers representing the rectangular echo area, in device coordinates. The echo area defines a portion of the workstation surface on which the prompt moves.

❸ The function GKS$INQ_LOCATOR_STATE initializes the variables you need to pass to the input functions. The argument GKS$K_VALUE_REALIZED tells the graphics handler to return the input values as they are implemented, instead of the way that the application may have set the values (GKS$K_VALUE_SET).

After the function call, RECORD_BUFFER_LENGTH contains the amount of the buffer filled with the written data record. If RECORD_SIZE is larger than RECORD_BUFFER_LENGTH, then you know that the data record was truncated to fit into your declared buffer.

❹ This code assigns new values to the input variables. For instance, the prompt and echo type is set to the number 1.

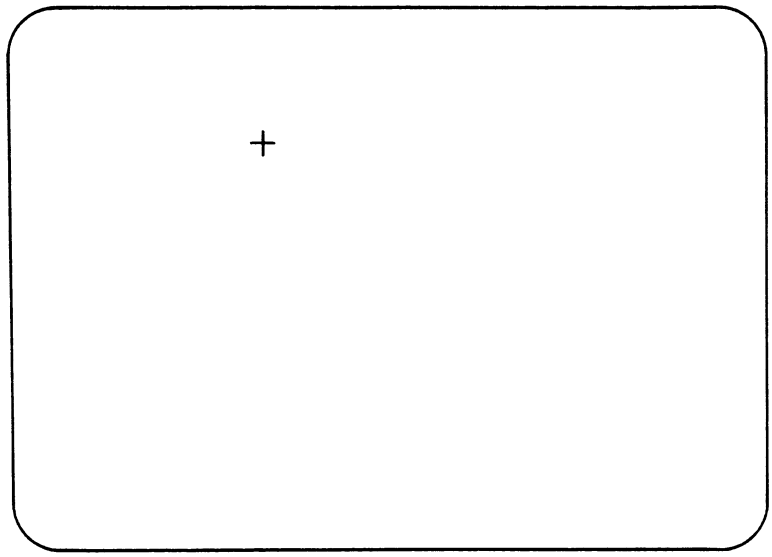
❺ The function GKS$INIT_LOCATOR initializes the locator logical input device.

❻ The function GKS$SET_LOCATOR_MODE sets the input operating mode to request and enables echo. Request input and enabled echo are the DEC GKS defaults.

❼ The function GKS$REQUEST_LOCATOR causes the device handler to place the input prompt on the workstation surface. At this point in the application, the input process takes control and the user can enter input. The device handler writes the world coordinate values, and the normalization transformation number used to translate the points, to the last arguments.

Figure 8–2 shows the screen of a VT241 terminal at the request for input.

Figure 8–2: Initializing the Locator Logical Input Device—VT241



ZK-5086-86

8.5.2 Sample Mode

In sample mode, the application program and the input process operate asynchronously. The user changes the input measure, and when the application chooses, it samples (takes) the current measure of the logical input device. The application determines when to end the input session.

As soon as you specify sample mode to one of the functions GKS$SET_LOCATOR_MODE, GKS$SET_STROKE_MODE, and so forth, the input prompt appears on the workstation surface (if echoing is enabled). At this point, the user can enter input, but cannot trigger the device or cancel input.

After you place the device in sample mode, you cannot reinitialize the device (by calling one of the GKS$INIT_class functions) until you remove the device's prompt from the workstation surface. To do this, place the device in request mode, reinitialize the device, and then place the device back into sample mode.

If you choose, you can place any or all of the supported logical input devices into sample mode at one time, but you only sample from one device at a time. At any point in the application, the program can call one of the functions GKS$SAMPLE_LOCATOR, GKS$SAMPLE_STROKE, and so forth, and the device handler returns the current measure from the specified workstation and the specified logical input device. When the program reaches some application-defined condition, the application can remove the input prompt from the workstation surface by changing the input mode from sample mode back to request mode.

When sampling choice and pick logical input devices, you can obtain an additional input status called GKS$K_STATUS_NOCHOICE (if the user did not alter the device's measure since it had been activated) or GKS$K_STATUS_NOPICK (if the user did not move the aperture, or if the user is not currently positioning the aperture on a segment). Under the specified conditions, DEC GKS writes these values to the input status argument.

Example 8–2 illustrates the use of the locator logical input device in sample mode. Following the program example, Figures 8–3 through 8–5 illustrate the program's effect on a VT241 workstation.

Example 8–2: Using a Locator Logical Input Device in Sample Mode

```
C     This program initializes and samples locator input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
❶     INTEGER WS_ID, DATA_RECORD( 1 ), PROMPT_ECHO_TYPE,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, XFORM,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE, INPUT_STATUS,
     * DEVICE_NUM
❷     REAL ECHO_AREA( 4 ), WORLD_X, WORLD_Y, LARGER
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, LARGER / 0.03 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT,
     * GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❸     CALL GKS$INQ_LOCATOR_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE,
     * ECHO_FLAG, XFORM, WORLD_X, WORLD_Y,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

C     Set the initial position of the cursor.
      WORLD_X = 0.9
      WORLD_Y = 0.0
```

(continued on next page)

Example 8–2 (Cont.): Using a Locator Logical Input Device in Sample Mode

```
C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

❹     CALL GKS$INIT_LOCATOR( WS_ID, DEVICE_NUM, WORLD_X,
     * WORLD_Y, XFORM, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH )

❺     CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_SAMPLE, GKS$K_ECHO )

C     Instruct the user.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.05, 0.95, 'Move the locator upwards.' )
      CALL GKS$TEXT( 0.05, 0.90, 'I will say when to stop.')

C     Do until the user moves the cursor closest to the top of the
C     device surface.
      DO WHILE ( WORLD_Y .LT. 0.9 )

❻     CALL GKS$SAMPLE_LOCATOR( WS_ID, DEVICE_NUM,
     * XFORM, WORLD_X, WORLD_Y )

C     Tease the user as the prompt gets closer.
      IF ( ( WORLD_Y .GT. 0.1 ) .AND.
     *     ( WORLD_Y .LT. 0.5 ) ) THEN
           CALL GKS$TEXT( 0.05, 0.85, 'You are still far away.')
      ENDIF
      IF ( ( WORLD_Y .GT. 0.5 ) .AND.
     *     ( WORLD_Y .LT. 0.7 ) ) THEN
           CALL GKS$TEXT( 0.05, 0.80, 'You are getting closer.')
      ENDIF
      IF ( ( WORLD_Y .GT. 0.7 ) .AND.
     *     ( WORLD_Y .LT. 0.9 ) ) THEN
           CALL GKS$TEXT( 0.05, 0.75, 'You are REALLY close.')
      ENDIF

      ENDDO

      CALL GKS$TEXT( 0.05, 0.70, 'YOU MADE IT!!!!')

C     Turn off the sample prompt.
❼     CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )
```

(continued on next page)

Example 8–2 (Cont.): Using a Locator Logical Input Device in Sample Mode

```
CALL GKS$DEACTIVATE_WS( WS_ID )
CALL GKS$CLOSE_WS( WS_ID )
CALL GKS$CLOSE_GKS()
END
```

The following numbers correspond to the numbers in the previous example:

❶ The DEC GKS VT241 handler ignores the data record for all supported locator prompt and echo types. This is a dummy argument.

❷ The echo area variable is an array of real numbers representing the rectangular echo area in device coordinates. The echo area defines a portion of the workstation surface on which the prompt moves.

❸ The function GKS$INQ_LOCATOR_STATE initializes the variables you need to pass to the input functions. The argument GKS$K_VALUE_REALIZED tells the graphics handler to return the input values as they are implemented, instead of the way that the application may have set the values (GKS$K_VALUE_SET).

After the function call, RECORD_BUFFER_LENGTH contains the amount of the buffer filled with the written data record. If RECORD_SIZE is larger than RECORD_BUFFER_LENGTH, then you know that the data record was truncated to fit into your declared buffer.

❹ The function GKS$INIT_LOCATOR initializes the locator logical input device.

❺ The function GKS$SET_LOCATOR_MODE sets the input operating mode to sample and enables echo.

❻ In the loop, the call to GKS$SAMPLE_LOCATOR writes the current measure of the device to its arguments until the user moves the cursor close to the top of the workstation surface. Note that when using a VT241 locator device number 1, the user moves the cursor using the arrow keys. Pressing RETURN has no effect in sample mode; the application has complete control in accepting a sample input value.

❼ When sampling is complete, the call to GKS$SET_LOCATOR_MODE sets the input operating mode to request mode, removes the prompt from the workstation surface, and ends the sample input session.

Figures 8–3 through 8–5 show the screen of a VT241 terminal as the user moves the cursor towards the top of the workstation surface. Notice that triggering the device does not affect the acceptance of input.

Figure 8–3: The Locator Logical Input Device in Sample Mode—VT241

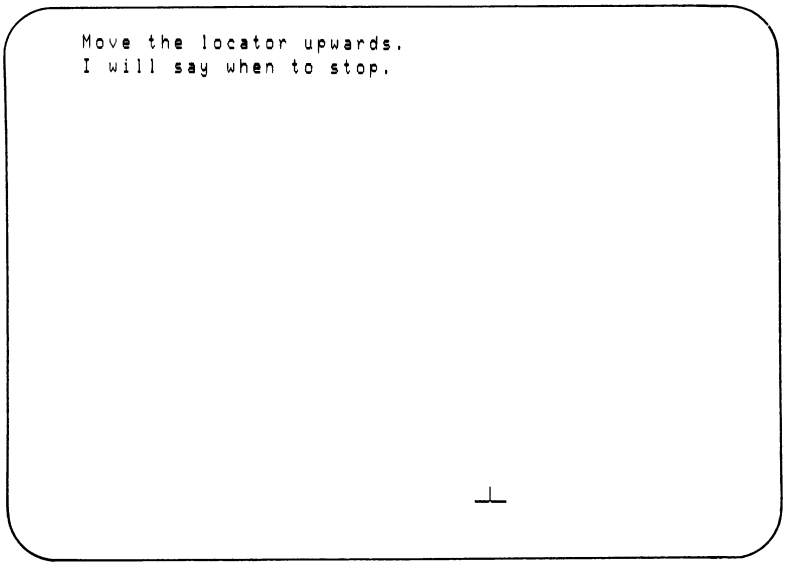
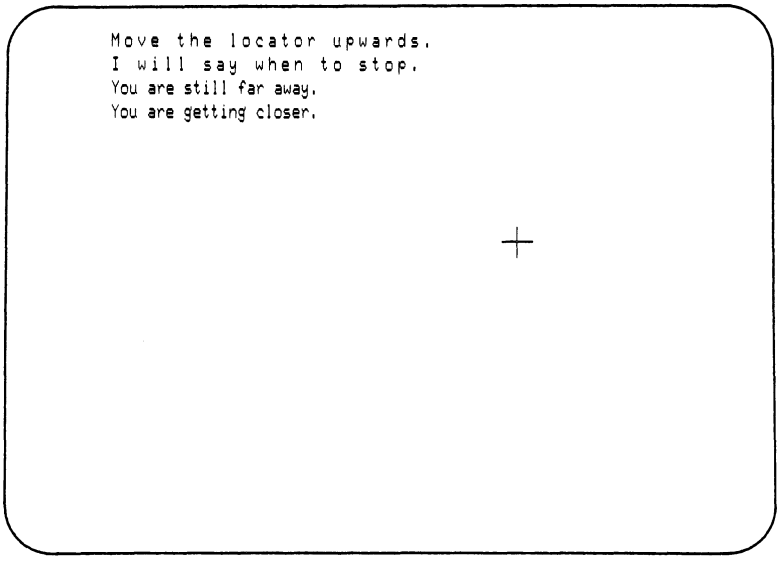
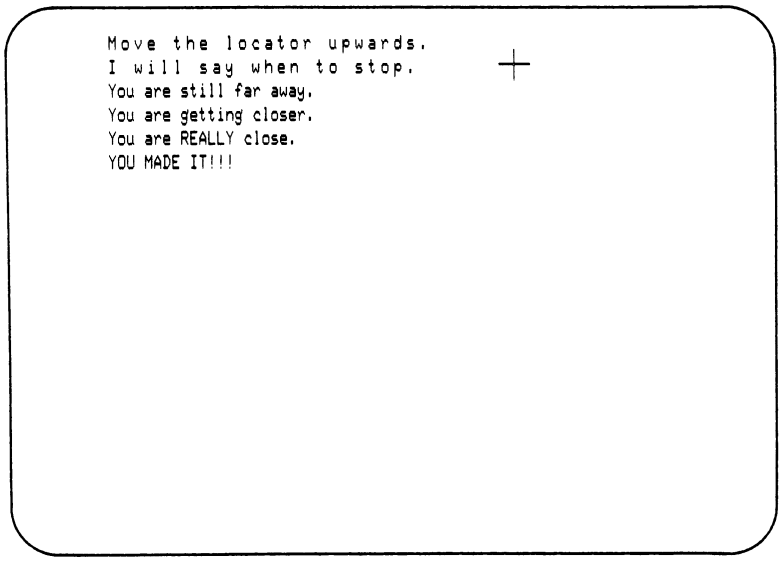


Figure 8–4: The Locator Logical Input Device in Sample Mode—VT241



ZK-5815-HC

Figure 8–5: The Locator Logical Input Device in Sample Mode—VT241



ZK-5816-HC

8.5.3 Event Mode

In event mode, the application program and the input process operate asynchronously. Event mode differs from sample mode in that the user must trigger input values that DEC GKS then places in a time-ordered queue. Each set of input values is a *report*. The application chooses when to remove the reports from the queue, beginning with the first input value entered by the user.

As soon as you specify event mode to one of the functions GKS$SET_LOCATOR_MODE, GKS$SET_STROKE_MODE, and so forth, the input prompt appears on the workstation surface (if echoing is enabled). At this point, the user can generate events that the device handler places in the event input queue.

After you place the device in event mode, you cannot reinitialize the device (by calling one of the GKS$INIT_class functions) until you remove the device's prompt from the workstation surface. To do this, place the device in request mode, reinitialize the device, and then place the device back into event mode.

When you choose, you can process reports generated by the user. To remove reports from the event input queue, call the function GKS$AWAIT_EVENT. GKS$AWAIT_EVENT checks the event queue, for a length of time up to the amount specified by the time-out argument. If the event queue contains at least one report, then GKS$AWAIT_EVENT removes the oldest report, places it in the *current event report* entry in the DEC GKS state list, and allows the application to resume. If the queue remains empty for the entire time-out period, GKS$AWAIT_EVENT writes GKS$K_INPUT_CLASS_NONE to its input class argument and allows the application to resume.

Each input report contains the following information that corresponds to the generated event:

- The workstation identifier
- The input class of the device
- The logical input device number

To process the information in the *current event report*, you need to check the value written to the input class argument of GKS$AWAIT_EVENT. Once you determine the class of the device that generated the event, you call one of the functions GKS$GET_LOCATOR, GKS$GET_STROKE, and so forth.

Remember that repeated calls to one of the functions GKS$GET_LOCATOR, GKS$GET_STROKE, and so forth, will write the same values to the output arguments since these funtions always obtain information from the current event report. The current event report does not change unless you call GKS$AWAIT_EVENT to fetch another report from the queue. Once you do this, a subsequent call to one of the GKS$GET_class functions obtains new input values.

If you decide that you have enough information from a particular logical input device, you can place the device back in request mode (stopping the generation of further events), and you can then flush all of the events generated by that device that remain in the event input queue. To flush reports generated by a logical input device from the event input queue, call the function GKS$FLUSH_DEVICE_EVENTS.

The following sections present the following information:

- A program example using event mode
- The handling of simultaneously generated events (GKS$K_INQ_MORE_SIMUL_EVENTS)
- The handling of input queue overflow (GKS$K_INQ_INPUT_QUEUE_OVERFLOW)

8.5.3.1 Program Example Using Event Mode

Example 8–3 illustrates the use of the locator logical input device in event mode. Following the program example, Figures 8–6 through 8–8 illustrate the program's effect on a VT241 workstation.

Example 8–3: Using a Locator Logical Input Device in Event Mode

```
C     This program initializes and generates locator events from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
(1)   INTEGER WS_ID, DATA_RECORD( 1 ), PROMPT_ECHO_TYPE,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, XFORM,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE, INPUT_STATUS,
     * DEVICE_NUM, CLASS
(2)   REAL ECHO_AREA( 4 ), WORLD_X, WORLD_Y, LARGER
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, LARGER / 0.03 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT,
     * GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Obtain the current locator input values.
(3)   CALL GKS$INQ_LOCATOR_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE,
     * ECHO_FLAG, XFORM, WORLD_X, WORLD_Y,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

C     Set the initial position of the cursor.
      WORLD_X = 0.9
      WORLD_Y = 0.0

C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

(4)   CALL GKS$INIT_LOCATOR( WS_ID, DEVICE_NUM, WORLD_X,
     * WORLD_Y, XFORM, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH )

(5)   CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_EVENT, GKS$K_ECHO )

C     Inform the user.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.05, 0.95, 'Move the locator upwards.' )
      CALL GKS$TEXT( 0.05, 0.90, 'Trigger until I say to stop.')
```

(continued on next page)

Example 8–3 (Cont.): Using a Locator Logical Input Device in Event Mode

```
C     Do until the user moves the cursor closest to the top of the
C     device surface.
      DO WHILE ( WORLD_Y .LT. 0.9 )

C     Check the event queue.
❻     CALL GKS$AWAIT_EVENT( 0.0, WS_ID, CLASS, DEVICE_NUM )

      IF ( CLASS .NE. GKS$K_INPUT_CLASS_NONE ) THEN
❼        CALL GKS$GET_LOCATOR( XFORM, WORLD_X, WORLD_Y )
      ENDIF

C     Tease the user as the prompt gets closer.
      IF ( ( WORLD_Y .GT. 0.1 ) .AND.
     *     ( WORLD_Y .LT. 0.5 ) ) THEN
           CALL GKS$TEXT( 0.05, 0.85, 'You are still far away.')
      ENDIF
      IF ( ( WORLD_Y .GT. 0.5 ) .AND.
     *     ( WORLD_Y .LT. 0.7 ) ) THEN
           CALL GKS$TEXT( 0.05, 0.80, 'You are getting closer.')
      ENDIF
      IF ( ( WORLD_Y .GT. 0.7 ) .AND.
     *     ( WORLD_Y .LT. 0.9 ) ) THEN
           CALL GKS$TEXT( 0.05, 0.75, 'You are REALLY close.')
      ENDIF

      ENDDO

      CALL GKS$TEXT( 0.05, 0.70, 'YOU MADE IT!!!!')

C     Turn off the event prompt.
❽     CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The DEC GKS VT241 handler ignores the data record for all supported locator prompt and echo types. This is a dummy argument.

❷ The echo area variable is an array of real numbers representing the rectangular echo area, in device coordinates. The echo area defines a portion of the workstation surface on which the prompt moves.

❸ The function GKS$INQ_LOCATOR_STATE initializes the variables you need to pass to the input functions. The argument GKS$K_VALUE_REALIZED tells the graphics handler to pass the input values as they are implemented, instead of the way that the application may have set the values (GKS$K_VALUE_SET).

After the function call, RECORD_BUFFER_LENGTH contains the amount of the buffer filled with the written data record. If RECORD_SIZE is larger than RECORD_BUFFER_LENGTH, then you know that the data record was truncated to fit into your declared buffer.

❹ The function GKS$INIT_LOCATOR initializes the locator logical input device.

❺ The function GKS$SET_LOCATOR_MODE sets the input operating mode to event and enables echo.

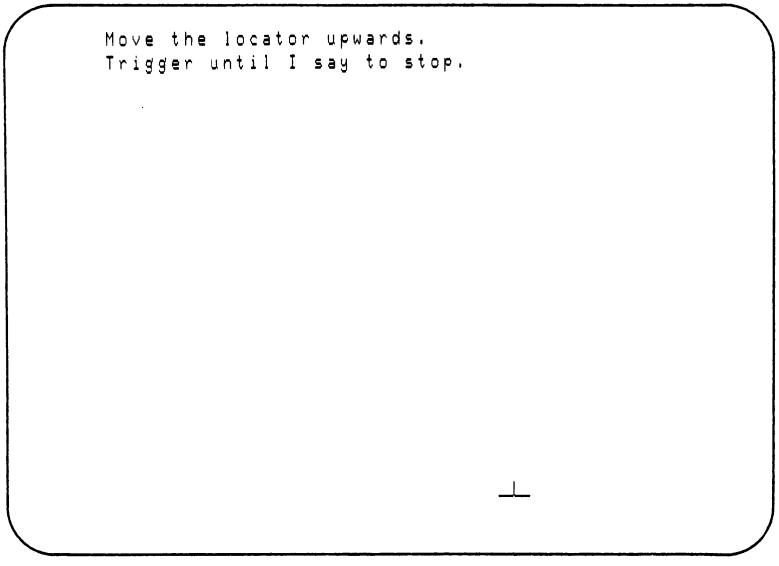
❻ In the loop, the call to GKS$AWAIT_EVENT immediately checks the input queue (as specified by the time-out argument of 0). If the user has not yet entered an event, or if the application has removed all reports generated thus far, GKS$AWAIT_EVENT returns GKS$K_INPUT_CLASS_NONE to its input class argument.

❼ As long as the input-class argument is not GKS$K_INPUT_CLASS_NONE, you can call GKS$GET_LOCATOR, since that is the only other device class that can generate an event.

❽ When accepting event reports is complete, the call to GKS$SET_LOCATOR_MODE sets the input operating mode to request mode, removes the prompt from the workstation surface, and ends the event input session.

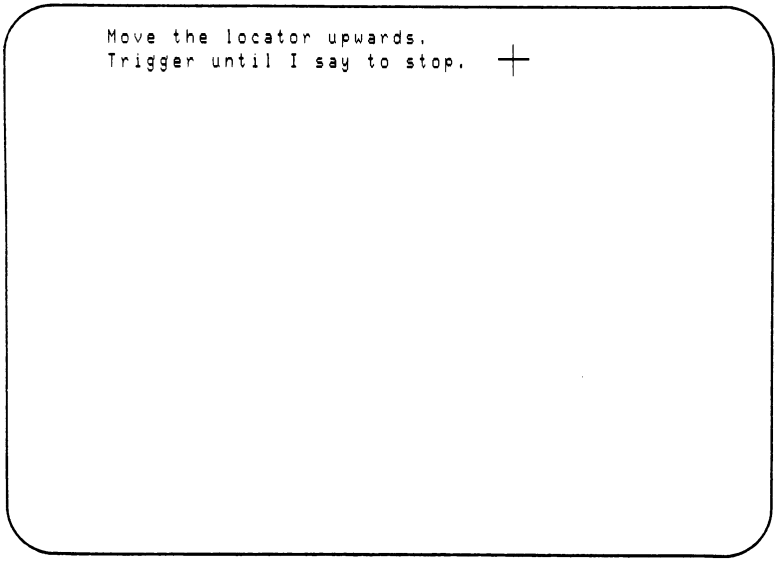
Figure 8–6 illustrates the surface of the VT241 when the input mode is set. Figure 8–7 illustrates the surface of the VT241 when the user moves the cursor but does *not* trigger the device. Figure 8–8 illustrates the surface of the VT241 when the user triggers an event.

Figure 8–6: The Locator Logical Input Device in Event Mode—VT241



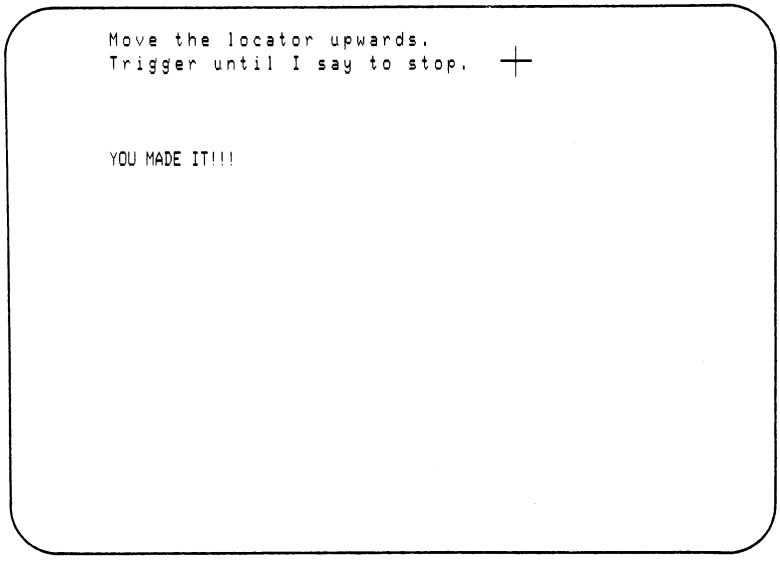
ZK-5834-HC

Figure 8–7: The Locator Logical Input Device in Event Mode—VT241



ZK-5835-HC

Figure 8–8: The Locator Logical Input Device in Event Mode—VT241



ZK-5837-HC

8.5.3.2 Placing Multiple Devices into Event Mode

This section describes the following:

- Methods by which you place more than one logical input device into event mode.
- How to handle simultaneous events.
- An example of cycling through devices currently in event mode.

Example 8–4 places both a pick and valuator logical input device into event mode, and handles simultaneously generated events. This example places two diagonally split boxes on the workstation's surface and defines them as separate segments. (This portion of the Example 8–7.) Using pick input to choose the appropriate box and valuator input to determine the amount of scaling, the user can generate events to increase or decrease the size of each box. Input ends when the user picks the triangle labeled STOP.

Following the program example, Figures 8–9 through 8–12 illustrate the effects of this program on a VT241.

Example 8–4: Placing Two Devices into Event Mode

```
C     This program generates events from two devices on a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, INITIAL_STATUS, SEGMENT,
     * PICK_ID, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE, PICK_INPUT_STATUS, DEVICE_NUM, BOX_1, BOX_2,
     * TRIANGLE_1, TRIANGLE_2, NUM_POINTS, CLASS
(1)   REAL ECHO_AREA( 4 ), PICK_DATA_RECORD( 1 ),
     * VAL_DATA_RECORD( 2 )
      REAL X_VALUES( 4 ), Y_VALUES( 4 ), LARGER,
(2)  * XFORM_MATRIX1( 6 ), XFORM_MATRIX2( 6 ), VALUE,
     * UPPER_LIMIT, LOWER_LIMIT
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, BOX_1 / 1 /,
     * BOX_2 / 2 /, TRIANGLE_1 / 1 /, TRIANGLE_2 / 2 /,
     * NUM_POINTS / 4 /, LARGER / 0.03 /
      DATA X_VALUES / 0.1, 0.4, 0.1, 0.1 /
      DATA Y_VALUES / 0.3, 0.6, 0.6, 0.3 /

C     The two elements in the valuator data record are the upper
C     and lower limits.
      EQUIVALENCE( VAL_DATA_RECORD( 1 ), LOWER_LIMIT )
      EQUIVALENCE( VAL_DATA_RECORD( 2 ), UPPER_LIMIT )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )

(3)   CALL GKS$CREATE_SEG( BOX_1 )
      CALL GKS$SET_PICK_ID( TRIANGLE_1 )
      CALL GKS$SET_FILL_COLOR_INDEX( 2 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      X_VALUES( 3 ) = 0.4
      Y_VALUES( 3 ) = 0.3
      CALL GKS$SET_PICK_ID( TRIANGLE_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 3 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      CALL GKS$CLOSE_SEG()
```

(continued on next page)

Example 8–4 (Cont.): Placing Two Devices into Event Mode

```
      X_VALUES( 1 ) = 0.6
      X_VALUES( 2 ) = 0.9
      X_VALUES( 3 ) = 0.6
      X_VALUES( 4 ) = 0.6
      Y_VALUES( 3 ) = 0.6

      CALL GKS$SET_PICK_ID( TRIANGLE_1 )

      CALL GKS$CREATE_SEG( BOX_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 2 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      X_VALUES( 3 ) = 0.9
      Y_VALUES( 3 ) = 0.3
      CALL GKS$SET_PICK_ID( TRIANGLE_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 3 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      CALL GKS$CLOSE_SEG()

      CALL GKS$SET_SEG_DETECTABILITY( BOX_1, GKS$K_DETECTABLE )
      CALL GKS$SET_SEG_DETECTABILITY( BOX_2, GKS$K_DETECTABLE )

      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.2, 0.45, '1')
      CALL GKS$TEXT( 0.3, 0.45, '2')
      CALL GKS$TEXT( 0.7, 0.45, '1')
      CALL GKS$TEXT( 0.8, 0.45, 'STOP')

C     Declare a data length of one long word which will hold the
C     size of the pick prompt.
(4)   RECORD_BUFFER_LENGTH = 4
      CALL GKS$INQ_PICK_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE,
     * ECHO_FLAG, INITIAL_STATUS, SEGMENT, PICK_ID,
     * PROMPT_ECHO_TYPE, ECHO_AREA, PICK_DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

      SEGMENT = BOX_1
      PICK_ID = TRIANGLE_1
      INITIAL_STATUS = GKS$K_STATUS_NOPICK
```

(continued on next page)

Example 8–4 (Cont.): Placing Two Devices into Event Mode

```
C     Initialize the segment transformation matrixes to the unity
C     transformtation.
      CALL GKS$EVAL_XFORM_MATRIX( 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
     * 1.0, GKS$K_COORDINATES_WC, XFORM_MATRIX1 )
      CALL GKS$EVAL_XFORM_MATRIX( 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
     * 1.0, GKS$K_COORDINATES_WC, XFORM_MATRIX2 )

C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_PICK( WS_ID, DEVICE_NUM, INITIAL_STATUS,
     * SEGMENT, PICK_ID, PROMPT_ECHO_TYPE, ECHO_AREA,
     * PICK_DATA_RECORD, RECORD_BUFFER_LENGTH )

C     Set the input operating mode to event.
      CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_EVENT, GKS$K_ECHO )

C     Reinitialize the input values for the valuator device.
❺     RECORD_BUFFER_LENGTH = 8
      CALL GKS$INQ_VALUATOR_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, VALUE,
     * PROMPT_ECHO_TYPE, ECHO_AREA, VAL_DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

      VALUE = 1.0
      UPPER_LIMIT = 1.5
      LOWER_LIMIT = 0.5

      CALL GKS$INIT_VALUATOR( WS_ID, DEVICE_NUM,
     * VALUE, PROMPT_ECHO_TYPE, ECHO_AREA,
     * VAL_DATA_RECORD, RECORD_BUFFER_LENGTH

      CALL GKS$SET_VALUATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_EVENT, GKS$K_ECHO )
```

(continued on next page)

Example 8–4 (Cont.): Placing Two Devices into Event Mode

```
C     Tell the user the task.
      CALL GKS$TEXT( 0.05, 0.95,
     * 'Move the cursor to scale a box.' )
      CALL GKS$TEXT( 0.05, 0.90,
     * 'PF1 cycles input devices.')
      CALL GKS$TEXT( 0.05, 0.85,
     * 'PF2 activates all devices.')
      CALL GKS$TEXT( 0.05, 0.80,
     * 'To finish, pick STOP.' )

C     Initialize variables.
      PICK_INPUT_STATUS = GKS$K_STATUS_NOPICK

C     Do until the user picks the second triangle in the second box.
❻     DO WHILE (( SEGMENT .NE. 2 ) .OR. ( PICK_ID .NE. 2 ))

C     Check the event queue.
      CALL GKS$AWAIT_EVENT( 0.0, WS_ID, CLASS, DEVICE_NUM )

C     Check for queue overflow.
❼     CALL GKS$INQ_INPUT_QUEUE_OVERFLOW( ERROR_STATUS, WS_ID,
     * CLASS, DEVICE_NUM )

C     If the queue has overflowed...
      IF ( ERROR_STATUS .EQ. 0 ) THEN
         CALL OVERFLOW( WS_ID, CLASS, DEVICE_NUM, PICK_INPUT_STATUS,
     *        SEGMENT, PICK_ID, XFORM_MATRIX1, XFORM_MATRIX2 )
      ENDIF

      CALL SCALE_IT( WS_ID, CLASS, DEVICE_NUM, PICK_INPUT_STATUS,
     *     SEGMENT, PICK_ID, XFORM_MATRIX1, XFORM_MATRIX2 )

      ENDDO

C     Turn off the event prompts.
      CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )
      CALL GKS$SET_VALUATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

(continued on next page)

Example 8–4 (Cont.): Placing Two Devices into Event Mode

```
C     ***************************************************************
C     Scale the appropriate segment...
      SUBROUTINE SCALE_IT( WS_ID, CLASS, DEVICE_NUM,
     * PICK_INPUT_STATUS, SEGMENT, PICK_ID, XFORM_MATRIX1,
     * XFORM_MATRIX2 )

      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
❽     INCLUDE 'SYS$LIBRARY:GKSMSGS.FOR'
      INTEGER CLASS, PICK_INPUT_STATUS, WS_ID, DEVICE_NUM,
     * PICK_ID, SEGMENT, MORE_EVENTS, ERROR_STATUS
      REAL VALUE, XFORM_MATRIX1( 6 ), XFORM_MATRIX2( 6 ), FIXED_X,
     * FIXED_Y, TEMP_X( 1 ), TEMP_Y( 1 )

C     Get all of the simultaneous event reports.
      MORE_EVENTS = GKS$K_MORE_EVENTS
❾     DO WHILE ( MORE_EVENTS .NE. GKS$K_NOMORE_EVENTS )

      IF ( CLASS .EQ. GKS$K_INPUT_CLASS_VALUATOR ) THEN
         CALL GKS$GET_VALUATOR( VALUE )
      ELSE IF ( CLASS .EQ. GKS$K_INPUT_CLASS_PICK ) THEN
         CALL GKS$GET_PICK( PICK_INPUT_STATUS, SEGMENT,
     *        PICK_ID )
      ENDIF

C     Set the flag MORE_EVENTS...
      CALL GKS$INQ_MORE_SIMUL_EVENTS( ERROR_STATUS,
     * MORE_EVENTS )

C     If there are more simultaneous events, take them from the queue...
      IF ( MORE_EVENTS .EQ. GKS$K_MORE_EVENTS) THEN
         CALL GKS$AWAIT_EVENT( 0.0, WS_ID, CLASS, DEVICE_NUM )
      ENDIF
      ENDDO

C     If appropriate, scale the segment.
❿     IF ((( PICK_INPUT_STATUS .NE. GKS$K_STATUS_NOPICK ) .AND.
     *    ( VALUE .NE. 1.0 )) .AND.
     *    ( CLASS .NE. GKS$K_INPUT_CLASS_NONE )) THEN
```

(continued on next page)

Example 8–4 (Cont.): Placing Two Devices into Event Mode

```
C     Scale the appropriate segment...
         IF ( SEGMENT .EQ. 1 ) THEN
            FIXED_X = 0.25
            FIXED_Y = 0.45
            CALL GKS$ACCUM_XFORM_MATRIX( XFORM_MATRIX1,
     *           FIXED_X, FIXED_Y, 0.0, 0.0, 0.0, VALUE,
     *           VALUE, GKS$K_COORDINATES_WC, XFORM_MATRIX1 )
            CALL GKS$SET_SEG_XFORM( SEGMENT, XFORM_MATRIX1 )
         ELSE
            FIXED_X = 0.75
            FIXED_Y = 0.45
            CALL GKS$ACCUM_XFORM_MATRIX( XFORM_MATRIX2,
     *           FIXED_X, FIXED_Y, 0.0, 0.0, 0.0, VALUE,
     *           VALUE, GKS$K_COORDINATES_WC, XFORM_MATRIX2 )
            CALL GKS$SET_SEG_XFORM( SEGMENT, XFORM_MATRIX2 )
         ENDIF

         CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      ENDIF

      END
```

The following numbers correspond to the numbers in the previous example:

❶ This application must define the two data records PICK_DATA_RECORD and VAL_DATA_RECORD. PICK_DATA_RECORD contains a single real value that is the size of the pick aperture in device coordinates, as required by the VT241 graphics handler. VAL_DATA_RECORD contains two real values that are the upper and lower limits of the real values.

❷ This code declares two separate segment transformation matrixes. These matrixes are required for scaling the boxes.

❸ The next lines of code draw and label each of the boxes. The two boxes are segments 1 and 2. The upper triangle within each box has a pick identifier of 1, and the other triangles have a pick identifier of 2.

❹ This code initializes the pick logical input device with the requested input values.

❺ This code initializes the valuator logical input device with the requested input values. The upper limit of the real values is 1.5, which increases the size of the box by 50 percent. The lower limit is 0.5, which decreases the size of the box by 50 percent.

❻ The WHILE loop calls the SCALE_IT subroutine until the user picks the second pick identifier (the lower triangle labelled STOP) in the second segment.

❼ This code calls the OVERFLOW subroutine in the event of an event input queue overflow. Section 8.5.3.3 describes this subroutine in detail.

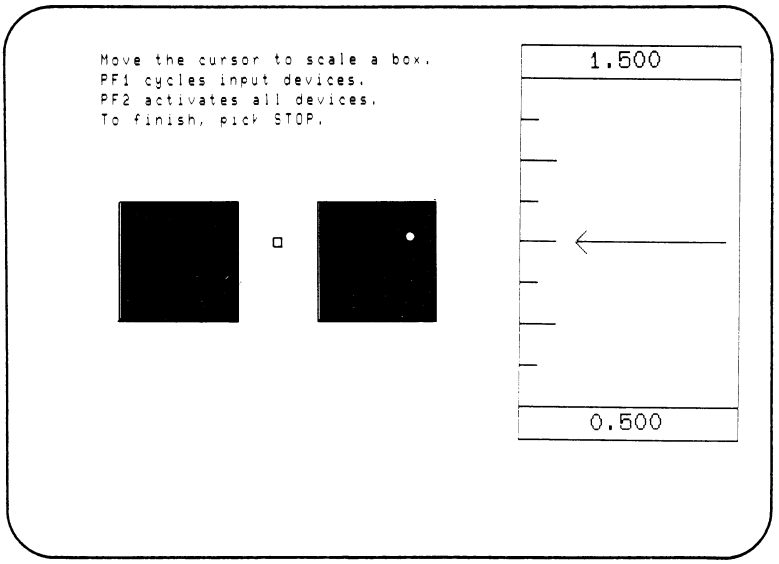
❽ The file GKSMSGS.FOR contains the constants that correspond with the DEC GKS error messages. The application includes this file to check for an error caused by the event input queue overflow.

❾ This WHILE loop gets information from the *current event report* depending on the value of the input class argument established in the previous call to GKS$AWAIT_EVENT. The loop checks for simultaneous events (multiple events entered in the queue by a single firing of a trigger) using a call to GKS$INQ_MORE_SIMUL_EVENTS. If more simultaneous events exist, this code calls GKS$AWAIT_EVENT to remove the event from the queue and process the information according to the input class of the device that generated the event. This loop continues until there are no more simultaneous events.

❿ This code checks to make sure that the input values would make a change to the current picture. (For instance, if the scaling factor is 1.0, then the application does not scale the segment.) If the values are appropriate, this code creates the corresponding segment transformation matrix and then scales the segment.

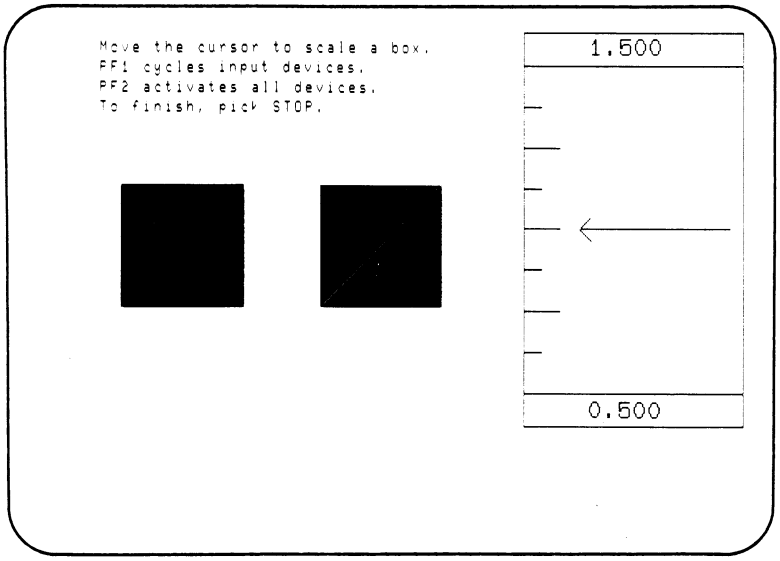
Figure 8–9 illustrates the effects of the program example before the user triggers any input; both of the logical input devices display their prompts. Figure 8–10 shows the workstation surface if the user presses the PF1 key; only the valuator device is active and displaying a prompt. DEC GKS determines in which order to place the devices in the input cycle. Figure 8–11 shows the workstation surface if the user presses the PF1 key again; only the pick device is active and displaying a prompt. Figure 8–12 shows the effects of generating events on both devices. Notice that the user has control over which input values are accepted (placed on the input queue by triggering the device); using sample mode, the application program has control over which input values are accepted (at the time of sampling).

Figure 8–9: Placing Two Devices in Event Mode—VT241



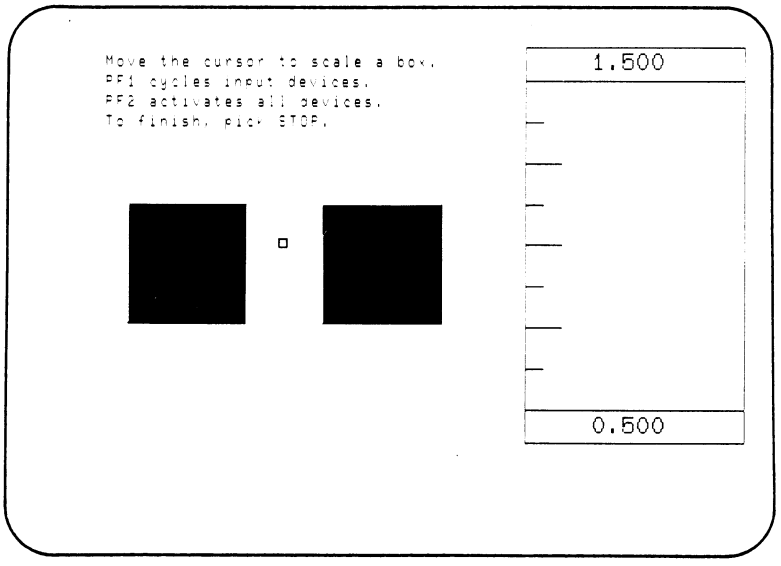
ZK-5817-HC

Figure 8–10: Placing Two Devices in Event Mode—VT241



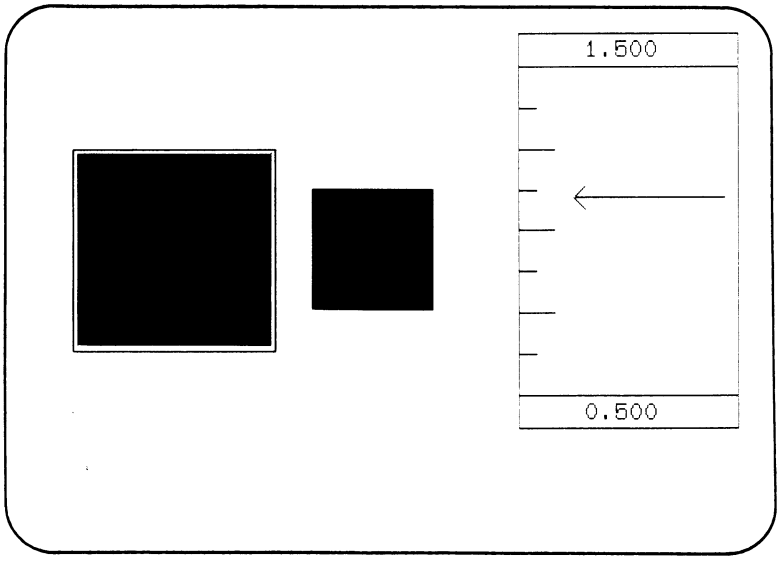
ZK-5818-HC

Figure 8–11: Placing Two Devices in Event Mode—VT241



ZK-5812-HC

Figure 8–12: Placing Two Devices in Event Mode—VT241



8.5.3.3 Event Input Queue Overflow

Since the user can generate events as soon as you call one of the functions GKS$SET_LOCATOR_MODE, GKS$SET_STROKE_MODE, and so forth, it is possible for the user to fill the event input queue before the application has the chance to remove any of the event reports.

If you attempt to call either GKS$AWAIT_EVENT or GKS$FLUSH_EVENTS in this situation, then DEC GKS logs an initial event input queue overflow error (GKS$K_ERROR_147—Input queue has overflowed). If you continue to call either GKS$AWAIT_EVENT or GKS$FLUSH_EVENTS, the functions still perform their task but the logical input devices cannot accept additional input until you clear the input queue. Since you can generate error GKS$K_ERROR_147 many times while attempting to clear the queue, DEC GKS logs the error only once, the first time it occurs.

To test for input queue overflow, you can call the function GKS$INQ_INPUT_QUEUE_OVERFLOW immediately after a call to GKS$AWAIT_EVENT. If the error status argument to GKS$INQ_INPUT_QUEUE_OVERFLOW is equal to the value 0, then the following is true.

- The event input queue has overflowed.
- Information about the overflow is available.
- GKS$INQ_INPUT_QUEUE_OVERFLOW writes the workstation identifier, the input class, and the logical device number of the device that last accepted input to its output arguments.

If the error status argument to GKS$INQ_INPUT_QUEUE_OVERFLOW is not equal to the value 0, then the information needed to write to the output arguments is not available. In this case, the error status argument can equal one of the following values.

- GKS$K_ERROR_7—GKS not in proper state.
- GKS$K_ERROR_148—Input queue has not overflowed since GKS was opened or since the last invocation of INQUIRE INPUT QUEUE OVERFLOW.
- GKS$K_ERROR_149—Input queue has overflowed, but the associated workstation has been closed.

If the event input queue overflows, you can continue to call GKS$AWAIT_EVENT, removing the reports one by one until a call returns GKS$K_INPUT_CLASS_NONE. As a second option, you can call the function GKS$FLUSH_DEVICE_EVENTS to remove the remaining reports generated by a device of a single input class. By calling GKS$FLUSH_DEVICE_EVENTS for all possible logical input classes, you clear the buffer and allow the user to enter input again.

Example 8–5 presents one method of handling an event input queue overflow.

Example 8–5: Subroutine Handling Event Queue Overflow

```
            .
            .
            .
C     Check for queue overflow.
      CALL GKS$INQ_INPUT_QUEUE_OVERFLOW( ERROR_STATUS, WS_ID,
     * CLASS, DEVICE_NUM )

C     If the event input queue has overflowed...
(1)   IF ( ERROR_STATUS .EQ. 0 ) THEN
         CALL OVERFLOW( WS_ID, CLASS, DEVICE_NUM, PICK_INPUT_STATUS,
     *        SEGMENT, PICK_ID, XFORM_MATRIX1, XFORM_MATRIX2 )
      ENDIF
            .
            .
            .
C     ************************************************************
C     Take care of the input queue overflow...
      SUBROUTINE OVERFLOW( WS_ID, CLASS, DEVICE_NUM,
     * PICK_INPUT_STATUS, SEGMENT, PICK_ID, XFORM_MATRIX1,
     * XFORM_MATRIX2 )

      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER CLASS, PICK_INPUT_STATUS, WS_ID, DEVICE_NUM,
     * PICK_ID, SEGMENT
      REAL VALUE, XFORM_MATRIX1( 6 ), XFORM_MATRIX2( 6 )

C     Stop any further input.
(2)   CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )
      CALL GKS$SET_VALUATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

(3)   DO WHILE (( CLASS .NE. GKS$K_INPUT_CLASS_NONE) .AND.
     *         ( CLASS .NE. GKS$K_INPUT_CLASS_PICK ))

C     Check the event queue.
      CALL SCALE_IT( WS_ID, CLASS, DEVICE_NUM, PICK_INPUT_STATUS,
     * SEGMENT, PICK_ID, XFORM_MATRIX1, XFORM_MATRIX2 )

      ENDDO
```

(continued on next page)

Example 8–5 (Cont.): Subroutine Handling Event Queue Overflow

```
C     If there is a pick event left, read the report.
      IF ( CLASS .EQ. GKS$K_INPUT_CLASS_PICK ) THEN
         CALL GKS$GET_PICK( PICK_INPUT_STATUS, SEGMENT,
     *        PICK_ID )
      ENDIF

C     Flush the queue of all event reports.
❹     CALL GKS$FLUSH_DEVICE_EVENTS( WS_ID,
     * GKS$K_INPUT_CLASS_VALUATOR, DEVICE_NUM )
      CALL GKS$FLUSH_DEVICE_EVENTS( WS_ID,
     * GKS$K_INPUT_CLASS_PICK, DEVICE_NUM )

C     Notify the user and return to event mode...
❺     CALL GKS$REDRAW_SEG_ON_WS( WS_ID )
      CALL GKS$TEXT( 0.05, 0.95, 'You entered input too fast!' )
      CALL GKS$TEXT( 0.05, 0.90, 'Please enter the input again.' )
      CALL GKS$TEXT( 0.05, 0.85, 'Press RETURN when ready.' )
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

      CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_EVENT, GKS$K_ECHO )
      CALL GKS$SET_VALUATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_EVENT, GKS$K_ECHO )

      END
          .
          .
          .
```

The following numbers correspond to the numbers in the previous example:

❶ If the error status argument to GKS$INQ_INPUT_QUEUE_OVERFLOW is the value 0, then there has been an overflow.

❷ You can stop the further generation of events by placing all logical input devices in request mode, thus removing the prompts from the workstation surface.

❸ This code removes each of the reports and scales the appropriate segment until it reaches a pick event or the end of the queue. The effect is to obtain the next pick event report, if one exists on the queue.

❹ You can use GKS$FLUSH_DEVICE_EVENTS to remove all reports remaining in the queue.

❺ This code notifies the user as to the cause of the delay in processing and then places the devices back into event mode. At this point, the user can once again generate event reports.

8.6 Function Descriptions

This section describes the DEC GKS input functions in detail.

Initializing Input

This section describes the functions used to specify input values to the logical input devices. In order to initialize a logical input device, you need to make sure that the device's prompt is not currently present on the surface of the workstation. So, to initialize a device, make sure that the device is in request mode (GKS$K_INPUT_MODE_REQUEST). This is the DEC GKS default mode.

If the device is in request mode and you activate an input device without first calling one of the GKS$INIT_class functions, the logical input device uses the device's default values (see Section 8.3.2 for more information).

This section describes the following functions:

- GKS$INIT_CHOICE
- GKS$INIT_LOCATOR
- GKS$INIT_PICK
- GKS$INIT_STRING
- GKS$INIT_STROKE
- GKS$INIT_VALUATOR

INITIALIZE CHOICE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$INIT_CHOICE establishes the initial values of a choice class device only if the device's prompt is not currently present on the workstation surface (the device must be in request mode). The initial values include the initial choice value, the prompt and echo type, the echo area, and the data record. Subsequent requests for choice input use the values you specify.

Syntax

GKS$INITIALIZE CHOICE *(workstation_id, device_number, initial_status, initial_choice, prompt_echo_type, echo_area, data_record, size_of_record)*

GINCH *(workstation_id, dev_num, in_status, in_choice, p_e_type, x_min, x_max, y_min, y_max, dim_dr, dr)*

ginitchoice *(workstation_id, dev, init, pet, area, record)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

initial_status

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the initial status of the logical input device. This argument determines whether a value is returned if the user triggers the request before moving the cursor. The argument can be either of the following constants or values:

Value	Constant	Description
1	GKS$K_STATUS_OK	The initial choice is chosen.
2	GKS$K_STATUS_NOCHOICE	No value is returned.

initial_choice

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the integer representing the initial highlighted choice.

prompt_echo_type

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the prompt and echo type.

echo_area

data type:	**array (real)**
access:	**read-only**
mechanism:	**by reference**

This argument is the echo area, which is a four-element array specifying the area on the workstation surface on which the prompt appears. Pass the device coordinates in the order X_MINIMUM, X_MAXIMUM, Y_MINIMUM, Y_MAXIMUM.

data_record

data type:	**address (record)**
access:	**read-only**
mechanism:	**by reference**

This argument is a pointer to the data record whose size and contents are dependent on the prompt and echo type, and on the graphics handler requirements. Each workstation may require a different data record structure with different contents.

size_of_record

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the amount of the data record buffer containing the actual data record, in bytes.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-90	DECGKS$_ERROR_NEG_90	Internal GKS error: Bad memory freed in routine ****
-93	DECGKS$_ERROR_NEG_93	Internal GKS error: Prompt and echo type not supported in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
46	GKS$_ERROR_46	Contents of input data record are invalid in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
122	GKS$_ERROR_122	Specified segment does not exist in routine ****
123	GKS$_ERROR_123	Specified segment does not exist on specified workstation in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****

Error Number	Completion Status Code	Message
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****
144	GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation in routine ****
145	GKS$_ERROR_145	Echo area is outside display space in routine ****
146	GKS$_ERROR_146	Contents of input data record are invalid in routine ****
152	GKS$_ERROR_152	Initial value is invalid in routine ****

Program Example

Example 8–6 illustrates the use of the function GKS$INIT_CHOICE. Following the program example, Figure 8–13 illustrates the program's effect on a VT241 workstation.

Example 8–6: Using a Choice Logical Input Device in Request Mode

```
C     This program initializes and requests choice input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
(1)   INTEGER WS_ID, DATA_RECORD( 3 ), NUM_CHOICES, SIZES( 3 ),
     * ADDRESSES( 3 ), PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH, RECORD_SIZE,
     * INPUT_STATUS, INITIAL_CHOICE, DEVICE_NUM, INPUT_CHOICE,
     * INITIAL_STATUS
(2)   REAL ECHO_AREA( 4 )
(3)   CHARACTER*80 CURRENT_STRINGS( 3 )
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /
```

(continued on next page)

Example 8–6 (Cont.): Using a Choice Logical Input Device in Request Mode

```
C     First element in the data record is the number of choices.
      EQUIVALENCE( DATA_RECORD(1), NUM_CHOICES )
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
C     Establish the size of the record buffer: 12 bytes.
      RECORD_BUFFER_LENGTH = 12

C     The second element in the VT241 choice data record is the pointer to
C     the array containing sizes of each choice character string. You need
C     to initialize the pointer so that the array can be initialized.
❹     DATA_RECORD( 2 ) = %LOC( SIZES(1) )

C     The third element in the VT241 choice data record is the pointer to the
C     array containing the pointers to the strings to be used. You need
C     to initialize the pointer so that the array can be initialized.
      DATA_RECORD( 3 ) = %LOC( ADDRESSES(1) )
      ADDRESSES( 1 ) = %LOC( CURRENT_STRINGS( 1 ) )
      ADDRESSES( 2 ) = %LOC( CURRENT_STRINGS( 2 ) )
      ADDRESSES( 3 ) = %LOC( CURRENT_STRINGS( 3 ) )

C     Inquire about the default values.
❺     NUM_CHOICES = 3
❻     CALL GKS$INQ_CHOICE_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, INITIAL_STATUS,
     * INITIAL_CHOICE, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH, RECORD_SIZE )
```

(continued on next page)

Example 8–6 (Cont.): Using a Choice Logical Input Device in Request Mode

```
      PROMPT_ECHO_TYPE = 1
      INITIAL_CHOICE = 2

C     Establish sizes of prompt strings...
      SIZES( 1 ) = 6
      SIZES( 2 ) = 6
      SIZES( 3 ) = 7
C     Establish locations of prompt strings...
      ADDRESSES( 1 ) = %LOC( 'Castro' )
      ADDRESSES( 2 ) = %LOC( 'Street' )
      ADDRESSES( 3 ) = %LOC( 'Station' )
C     Since the device is in request mode by default, initialize the device.
❼     CALL GKS$INIT_CHOICE( WS_ID, DEVICE_NUM,
     * INITIAL_STATUS, INITIAL_CHOICE, PROMPT_ECHO_TYPE,
     * ECHO_AREA, DATA_RECORD, RECORD_BUFFER_LENGTH )

❽     CALL GKS$SET_CHOICE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

❾     CALL GKS$REQUEST_CHOICE( WS_ID, DEVICE_NUM, INPUT_STATUS,
     * INPUT_CHOICE )

C     Output the input choice number.
      WRITE(6,*) INPUT_CHOICE
      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ For a VT241 choice logical input device with a prompt and echo type 1, the data record contains three values: the number of choices, the address of an array containing the size of the prompt strings, and the address of an array containing the addresses of the prompt strings.

❷ The echo area variable is an array of real numbers representing the rectangular echo area, in device coordinates. The echo area defines a portion of the workstation surface on which the choice appears.

❸ The program uses this array to store the current choice strings. If you do not initialize the elements in the array ADDRESSES to point to the elements in CURRENT_STRINGS, the function GKS$INQ_CHOICE_STATE does not have a buffer in which to write the current strings.

❹ Before calling GKS$INQ_CHOICE_STATE, you must initialize three modifiable arguments: RECORD_BUFFER_LENGTH, DATA_RECORD(2), DATA_RECORD(3), and the buffers for the current choice strings. The argument RECORD_BUFFER_LENGTH tells DEC GKS the size of the buffer. The two components of DATA_RECORD contain addresses that tell DEC GKS where to write the initial string addresses and the initial strings. If you do not initialize these three modifiable arguments, you generate an error.

❺ This code sets the number of choices to be the number 3.

❻ The function GKS$INQ_CHOICE_STATE initializes the variables you need to pass to the input functions. After the function call, RECORD_BUFFER_LENGTH contains the amount of the buffer filled with the written data record. If RECORD_SIZE is larger than RECORD_BUFFER_LENGTH, then you know that the data record was truncated to fit into your declared buffer.

❼ The function GKS$INIT_CHOICE initializes the choice logical input device.

❽ The call to GKS$SET_CHOICE_MODE places the logical input device into request mode and enables echoing of the input.

❾ The function GKS$REQUEST_CHOICE prompts the user for input. The integral choice is written to the last argument.

Figure 8–13 shows the screen of a VT241 terminal at the request for input.

Figure 8–13: Requesting Input from a Choice Logical Input Device—VT241

INITIALIZE LOCATOR

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$INIT_LOCATOR establishes the initial values of a locator class logical input device only if the device's prompt is not currently present on the workstation surface (the device must be in request mode). The initial values include the world coordinates of the initial locator, the normalization transformation used to transform the initial locator point, the prompt and echo type, the echo area, and the data record. Subsequent requests for locator input use the values you specify.

For more information about the locator position and echo types 2 and 3, see the Chapter 1, VAXstation Workstation Specifics, in the *DEC GKS Device Specifics Reference Manual.*

If you do not call GKS$INIT_LOCATOR before you request input from a locator logical input device, DEC GKS uses the default input values.

Syntax

GKS$INIT_LOCATOR *(workstation_id, device_number, initial_x_value, initial_y_value, transformation_number, prompt_echo_type, echo_area, data_record, size_of_record)*

GINLC *(workstation_id, dev_num, x_form, px, py, p_e_type, x_min, x_max, y_min, y_max, dim_dr, dr)*

ginitloc *(workstation_id,dev, init, pet, area, record)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

initial_x_value
initial_y_value

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments are the real numbers representing the initial position of the prompt, in world coordinates.

transformation_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the normalization transformation number used to transform the initial point from world coordinates to normalized device coordinates.

prompt_echo_type

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the prompt and echo type.

echo_area

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

This argument is the echo area, which is a four-element array specifying the area on the workstation surface on which the prompt appears. Pass the device coordinates in the order X_MINIMUM, X_MAXIMUM, Y_MINIMUM, Y_MAXIMUM.

data_record

data type: **address (record)**
access: **read-only**
mechanism: **by reference**

This argument is a pointer to the data record whose size and contents are dependent on the prompt and echo type, and on the graphics handler requirements. Each workstation may require a different data record structure with different contents.

size_of_record

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the amount of the data record buffer containing the actual data record, in bytes.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-93	DECGKS$_ERROR_NEG_93	Internal GKS error: Prompt and echo type not supported in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
46	GKS$_ERROR_46	Contents of input data record are invalid in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****
60	GKS$_ERROR_60	Polyline index is invalid in routine ****
63	GKS$_ERROR_63	Linetype is equal to zero in routine ****
65	GKS$_ERROR_65	Linewidth scale factor is less than zero in routine ****
80	GKS$_ERROR_80	Fill area index is invalid in routine ****
84	GKS$_ERROR_84	Style (pattern or hatch) index is equal to zero in routine ****
92	GKS$_ERROR_92	Color index is less than zero in routine ****

Error Number	Completion Status Code	Message
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****
144	GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation in routine ****
145	GKS$_ERROR_145	Echo area is outside display space in routine ****
152	GKS$_ERROR_152	Initial value is invalid in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–1.

INITIALIZE PICK

Operating States: WSOP, WSAC, SGOP

Description

This function establishes the initial values of a pick class logical input device only if the device's prompt is not currently present on the workstation surface (the device must be in request mode). The initial values include the initial status value, the initial segment, the initial pick identifier, the prompt and echo type, the echo area, and the data record. Subsequent requests for pick input use the values you specify.

If you do not call GKS$INIT_PICK before you request input from a pick logical input device, DEC GKS uses the default input values. For more information concerning the default input values, refer to the *DEC GKS Device Specifics Reference Manual.*

Syntax

GKS$INIT_PICK *(workstation_id, device_number, initial_status, initial_segment, initial_pick_id, echo_type, echo_area, data_record, size_of_record)*

GINPK *(workstation_id, dev_num, istatus, isegment, i_pick_id, p_e_type, x_min, x_max, y_min, y_max, dim_dr, dr)*

ginitchoice *(workstation_id,dev, init, pet, area, record)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class, operating on the same workstation.

initial_status

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the initial status of the logical input device. This argument determines whether a value is returned if the user triggers the request before moving the cursor. The argument can be either of the following constants or values:

Value	Constant	Description
1	GKS$K_STATUS_OK	The initial segment and pick identifier are chosen.
2	GKS$K_STATUS_NOPICK	No segment or pick identifier is returned.

initial_segment

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the segment identifier that tells DEC GKS on which segment to place the prompt initially.

initial_pick_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the pick identifier used to tell DEC GKS where in the chosen segment to place the prompt initially. A pick identifier is an integer that represents a portion of segment, allowing you to pick primitives instead of picking the entire segment.

echo_type

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the prompt and echo type.

echo_area

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

This argument is the echo area, which is a four-element array specifying the area on the workstation surface on which the prompt appears. Pass the device coordinates in the order X_MINIMUM, X_MAXIMUM, Y_MINIMUM, Y_MAXIMUM.

data_record

data type: **address (record)**
access: **read-only**
mechanism: **by reference**

This argument is a pointer to the data record whose size and contents are

dependent on the prompt and echo type, and on the graphics handler requirements. Each workstation may require a different data record structure with different contents.

size_of_record

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the amount of the data record buffer containing the actual data record, in bytes.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-93	DECGKS$_ERROR_NEG_93	Internal GKS error: Prompt and echo type not supported in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
37	GKS$_ERROR_37	Specified workstation is not of category OUTIN in routine ****
46	GKS$_ERROR_46	Contents of input data record are invalid in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****

Error Number	Completion Status Code	Message
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****
144	GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation in routine ****
145	GKS$_ERROR_145	Echo area is outside display space in routine ****
152	GKS$_ERROR_152	Initial value is invalid in routine ****

Program Example

Example 8–7 illustrates the use of the function GKS$INIT_PICK. Following the program example, Figure 8–14 illustrates the program's effect on a VT241 workstation.

Example 8–7: Using a Pick Logical Input Device in Request Mode

```
C     This program initializes and requests pick input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, INITIAL_STATUS, SEGMENT,
     * PICK_ID, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE, INPUT_STATUS,
     * DEVICE_NUM, BOX_1, BOX_2, TRIANGLE_1,
     * TRIANGLE_2, NUM_POINTS
❶     REAL ECHO_AREA(4), DATA_RECORD( 1 )
      REAL X_VALUES( 4 ), Y_VALUES( 4 )
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, BOX_1 / 1 /,
     * BOX_2 / 2 /, TRIANGLE_1 / 1 /, TRIANGLE_2 / 2 /,
     * NUM_POINTS / 4 /
      DATA X_VALUES / 0.1, 0.4, 0.1, 0.1 /
      DATA Y_VALUES / 0.3, 0.6, 0.6, 0.3 /
```

(continued on next page)

Example 8–7 (Cont.): Using a Pick Logical Input Device in Request Mode

```
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )

❷     CALL GKS$CREATE_SEG( BOX_1 )
      CALL GKS$SET_PICK_ID( TRIANGLE_1 )
      CALL GKS$SET_FILL_COLOR_INDEX( 2 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      X_VALUES( 3 ) = 0.4
      Y_VALUES( 3 ) = 0.3
      CALL GKS$SET_PICK_ID( TRIANGLE_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 3 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      CALL GKS$CLOSE_SEG()

❸     X_VALUES( 1 ) = 0.6
      X_VALUES( 2 ) = 0.9
      X_VALUES( 3 ) = 0.6
      X_VALUES( 4 ) = 0.6
      Y_VALUES( 3 ) = 0.6

      CALL GKS$SET_PICK_ID( TRIANGLE_1 )

❹     CALL GKS$CREATE_SEG( BOX_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 2 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      X_VALUES( 3 ) = 0.9
      Y_VALUES( 3 ) = 0.3
      CALL GKS$SET_PICK_ID( TRIANGLE_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 3 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      CALL GKS$CLOSE_SEG()

      CALL GKS$SET_SEG_DETECTABILITY( BOX_1, GKS$K_DETECTABLE )
      CALL GKS$SET_SEG_DETECTABILITY( BOX_2, GKS$K_DETECTABLE )

❺     CALL GKS$SET_TEXT_HEIGHT( 0.03 )
      CALL GKS$TEXT( 0.2, 0.45, '1')
      CALL GKS$TEXT( 0.3, 0.45, '2')
      CALL GKS$TEXT( 0.7, 0.45, '1')
      CALL GKS$TEXT( 0.8, 0.45, '2')
```

(continued on next page)

Example 8–7 (Cont.): Using a Pick Logical Input Device in Request Mode

```
C     Declare a data length of one long word which will hold the
C     size of the pick prompt.
❻     RECORD_BUFFER_LENGTH = 4
      CALL GKS$INQ_PICK_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE,
     * ECHO_FLAG, INITIAL_STATUS, SEGMENT, PICK_ID,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

❼     SEGMENT = BOX_1
      PICK_ID = TRIANGLE_1
      PROMPT_ECHO_TYPE = 1
      INITIAL_STATUS = GKS$K_STATUS_NOPICK

C     Since the device is in request mode by default, initialize the device.
❽     CALL GKS$INIT_PICK( WS_ID, DEVICE_NUM, INITIAL_STATUS,
     * SEGMENT, PICK_ID, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH )

❾     CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

❿     CALL GKS$REQUEST_PICK( WS_ID, DEVICE_NUM, INPUT_STATUS,
     * SEGMENT, PICK_ID )

C     Output the segment number and pick identifier.
      WRITE(6,*) INPUT_STATUS, SEGMENT, PICK_ID
      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The DEC GKS VT241 handler requires a data record for prompt and echo type 1 that contains a real value (size of the pick aperture prompt in device coordinates).

The echo area variable is an array of real numbers representing the rectangular echo area, in device coordinates. The echo area defines a portion of the workstation surface on which the prompt moves.

❷ This code creates a box on the left side of the workstation surface and places it in a segment. The code divides the box diagonally and sets pick identifiers for each of the created triangles.

❸ This code resets the X and Y world coordinate values so that the position of the box changes.

❹ This code creates a box on the right side of the workstation surface and places it in a segment. The code divides the box diagonally and sets pick identifiers for each of the created triangles.

❺ This code labels the triangles by their pick identifiers.

❻ The function GKS$INQ_PICK_STATE initializes the variables you need to pass to the input functions. The argument GKS$K_VALUE_REALIZED tells the graphics handler to pass the input values as they are implemented, as opposed to the way that the application may have set the values (GKS$K_VALUE_SET).

After the function call, RECORD_BUFFER_LENGTH contains the amount of the buffer filled with the written data record. If RECORD_SIZE is larger than RECORD_BUFFER_LENGTH, then you know that the data record was truncated to fit into your declared buffer.

❼ This code assigns new values to the input variables. For instance, the initial segment identifier has the value 1.

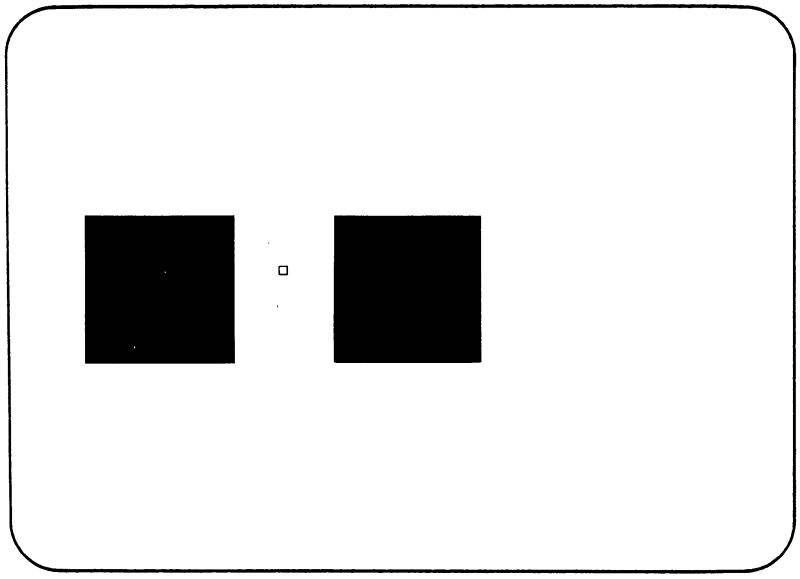
❽ The function GKS$INIT_PICK initializes the request for choice input.

❾ The call to GKS$SET_PICK_MODE places the logical input device into request mode and enables echoing of the input.

❿ The function GKS$REQUEST_PICK prompts the user for input. The segment and pick identifiers are written to the last arguments.

Figure 8–14 shows the screen of a VT241 terminal at the request for input.

Figure 8–14: Requesting Input from the Pick Input Device—VT241



ZK-5087-86

INITIALIZE STRING

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$INIT_STRING establishes the initial values of a string class logical input device only if the device's prompt is not currently present on the workstation surface (the device must be in request mode). The initial values include the initial string value, the prompt and echo type, the echo area, and the data record. Subsequent requests for choice input use the values you specify.

If you do not call GKS$INIT_STRING before you request input from the string logical input device, GKS uses the default input values.

Syntax

GKS$INIT_STRING *(workstation_id, device_number, initial_string, echo_type, echo_area, data_record, size_of_record)*

GINST *(workstation_id, dev_num, lstring, istring, p_e_type, x_min, x_max, y_min, y_max, buf_len, i_cur_pos, dim_dr, dr)*

GINST - Subset *(workstation_id, dev_num, lstring, istring, p_e_type, x_min, x_max, y_min, y_max, buf_len, i_cur_pos, dim_dr, dr)*

ginitstring *(workstation_id,dev, init, pet, area, record)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

initial_string

data type: **string**
access: **read-only**
mechanism: **by descriptor**

This argument is the initial string displayed on the workstation surface. Once you request input, the user can delete or edit the initial string; otherwise, the newly input string is appended to the initial string.

echo_type

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the prompt and echo type.

echo_area

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

This argument is the echo area, which is a four-element array specifying the area on the workstation surface on which the prompt appears. Pass the device coordinates in the order X_MINIMUM, X_MAXIMUM, Y_MINIMUM, Y_MAXIMUM.

data_record

data type: **address (record)**
access: **read-only**
mechanism: **by reference**

This argument is a pointer to the data record whose size and contents are dependent on the prompt and echo type, and on the graphics handler require-

ments. Each workstation may require a different data record structure with different contents. Refer to the *DEC GKS Device Specifics Reference Manual* for more information concerning the device's data record requirements.

size_of_record

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the amount of the data record buffer containing the actual data record, in bytes.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-34	DECGKS$_ERROR_NEG_34	String length less than or equal to 0 in routine ****
-93	DECGKS$_ERROR_NEG_93	Internal GKS error: Prompt and echo type not supported in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
46	GKS$_ERROR_46	Contents of input data record are invalid in routine ****

Error Number	Completion Status Code	Message
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****
144	GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation in routine ****
145	GKS$_ERROR_145	Echo area is outside display space in routine ****
152	GKS$_ERROR_152	Initial value is invalid in routine ****
154	GKS$_ERROR_154	Length of the initial string is greater than the buffer size in routine ****

Program Example

Example 8–8 illustrates the use of the function GKS$INIT_STRING. Following the program example, Figure 8–15 illustrates the program's effect on a VT241 workstation.

Example 8–8: Using a String Logical Input Device in Request Mode

```
C     This program initializes and requests string input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
❶     INTEGER WS_ID, DATA_RECORD( 2 ),
     * PROMPT_ECHO_TYPE, ERROR_STATUS, BUFFER_LENGTH,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE, INPUT_STATUS, DEVICE_NUM, STRING_SIZE,
     * CUR_POSITION
❷     REAL ECHO_AREA( 4 )
❸     CHARACTER*80 INITIAL_STRING, INPUT_STRING
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /

C     First element in the data record is length of the buffer that
C     contains the input string.
      EQUIVALENCE( DATA_RECORD( 1 ), BUFFER_LENGTH )
      EQUIVALENCE( DATA_RECORD( 2 ), CUR_POSITION )
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
      RECORD_BUFFER_LENGTH = 8
❹     CALL GKS$INQ_STRING_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, INITIAL_STRING,
     * STRING_SIZE, PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

❺     BUFFER_LENGTH = 15
      PROMPT_ECHO_TYPE = 1

C     Since the device is in request mode by default, initialize the device.
❻     CALL GKS$INIT_STRING( WS_ID, DEVICE_NUM, 'GKS>',
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )

❼     CALL GKS$SET_STRING_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )
```

(continued on next page)

Example 8–8 (Cont.): Using a String Logical Input Device in Request Mode

```
❽      CALL GKS$REQUEST_STRING( WS_ID, DEVICE_NUM, INPUT_STATUS,
      * INPUT_STRING, STRING_SIZE )

C      Output the input string and its size.
       WRITE(6,*) INPUT_STRING, STRING_SIZE
       CALL GKS$DEACTIVATE_WS( WS_ID )
       CALL GKS$CLOSE_WS( WS_ID )
       CALL GKS$CLOSE_GKS()
       END
```

The following numbers correspond to the numbers in the previous example:

❶ For all logical input prompt and echo types, the data record contains the length of the buffer and the initial editing position . This buffer can only be as large as the maximum size supported by the workstation. To obtain that size, you can call GKS$INQ_DEF_STRING_DATA.

❷ The echo area variable is an array of real numbers representing the rectangular echo area, in device coordinates. The echo area defines a portion of the workstation surface on which the initial string appears.

❸ The defined string variables can contain a string the length of the terminal screen. You can alter the maximum size of the input string, every time you initialize the string logical input device, by changing the value associated with the buffer length.

❹ The function GKS$INQ_STRING_STATE initializes the variables you need to pass to the input functions. After the function call, RECORD_BUFFER_LENGTH contains the amount of the buffer filled with the written data record. If RECORD_SIZE is larger than RECORD_BUFFER_LENGTH, then you know that the data record was truncated to fit into your declared buffer.

❺ This code assigns new values to the input variables. For instance, the buffer length is defined to be 15 bytes.

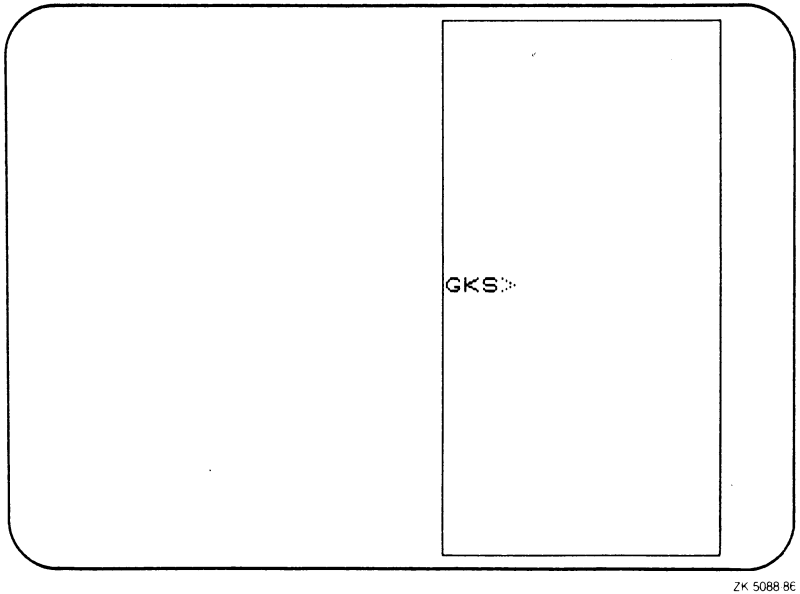
❻ The function GKS$INIT_STRING initializes the string logical input device.

❼ The call to GKS$SET_STRING_MODE places the logical input device into request mode and enables echoing of the input.

❽ The function GKS$REQUEST_STRING prompts the user for input. The input string is written to the second to last argument. The last argument contains the size of the input string.

Figure 8–15 shows the screen of a VT241 terminal at the request for input.

Figure 8–15: Requesting from the String Logical Input Device—VT241



ZK-5088-8E

INITIALIZE STROKE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$INIT_STROKE establishes the initial values of a stroke class logical input device only if the device's prompt is not currently present on the workstation surface (the device must be in request mode). The initial values include the number of points in the initial stroke, the world coordinate values in the initial stroke, the normalization transformation number used to translate world coordinates of the initial stroke to NDC points, the prompt and echo type, the echo area, and the data record. Subsequent requests for choice input use the values you specify.

If you do not call GKS$INIT_STROKE before you request input from a stroke logical input device, DEC GKS uses the default input values.

Syntax

GKS$INIT_STROKE *(workstation_id, device_number, initial_number_points, initial_stroke_x_values, initial_stroke_y_values, transformation_number, echo_type, echo_area, data_record, size_of_record)*

GINSK *(workstation_id, dev_num, xform, num_pts, px, py, p_e_type, x_min, x_max, y_min, y_max, buf_len, dim_dr, dr)*

ginitstroke *(workstation_id,dev, init, pet, area, record)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

initial_number_points

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the number of points in the initial stroke.

initial_stroke_x_values
initial_stroke_y_values

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

These arguments are the X and Y world coordinate values in the initial stroke.

transformation_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the normalization transformation number used to transform the initial stroke from world coordinates to normalized device coordinates.

echo_type

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the prompt and echo type.

echo_area

data type:	**(real)**
access:	**read-only**
mechanism:	**by reference**

This argument is the echo area, which is a four-element array specifying the area on the workstation surface on which the prompt appears. Pass the device coordinates in the order X_MINIMUM, X_MAXIMUM, Y_MINIMUM, Y_MAXIMUM.

data_record

data type:	**address (record)**
access:	**read-only**
mechanism:	**by reference**

This argument is a pointer to the data record whose size and contents are dependent on the prompt and echo type, and on the graphics handler requirements. Each workstation may require a different data record structure with different contents.

size_of_record

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the amount of the data record buffer containing the actual data record, in bytes.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-93	DECGKS$_ERROR_NEG_93	Internal GKS error: Prompt and echo type not supported in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
46	GKS$_ERROR_46	Contents of input data record are invalid in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****
60	GKS$_ERROR_60	Polyline index is invalid in routine ****
63	GKS$_ERROR_63	Linetype is equal to zero in routine ****
65	GKS$_ERROR_65	Linewidth scale factor is less than zero in routine ****
66	GKS$_ERROR_66	Polymarker index is invalid in routine ****
67	GKS$_ERROR_67	A representation for the specified polymarker index has been defined on this workstation in routine *** *

Error Number	Completion Status Code	Message
69	GKS$_ERROR_69	Marker type is equal to zero in routine ****
92	GKS$_ERROR_92	Color index is less than zero in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****
144	GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation in routine ****
145	GKS$_ERROR_145	Echo area is outside display space in routine ****
152	GKS$_ERROR_152	Initial value is invalid in routine ****
153	GKS$_ERROR_153	The number of points in the initial stroke is greater than the buffer size in routine ****

Program Example

Example 8–9 illustrates the use of the function GKS$INIT_STROKE. Following the program example, Figure 8–16 illustrates the program's effect on a VT241 workstation.

Example 8–9: Using a Stroke Logical Input Device in Request Mode

```
C     This program initializes and requests stroke input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
❶     INTEGER WS_ID, DATA_RECORD( 6 ), BUFFER_SIZE,
     * DIMENSION, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * TRANSFRM, NUM_POINTS, INPUT_MODE, ECHO_FLAG,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE, INPUT_STATUS, DEVICE_NUM,
     * RET_SIZE_X, RET_SIZE_Y, I, EDIT_POSITION, ATTS_FLAG
❷     REAL ECHO_AREA( 4 ), STROKE_X( 50 ),
     * STROKE_Y( 50 ), X_INT, Y_INT, TIME_INT
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /

C     First element in the data record is the buffer size.
      EQUIVALENCE( DATA_RECORD( 1 ), BUFFER_SIZE)
      EQUIVALENCE( DATA_RECORD( 2 ), EDIT_POSITION)
      EQUIVALENCE( DATA_RECORD( 3 ), X_INT)
      EQUIVALENCE( DATA_RECORD( 4 ), Y_INT)
      EQUIVALENCE( DATA_RECORD( 5 ), TIME_INT)
      EQUIVALENCE( DATA_RECORD( 6 ), ATTS_FLAG)

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
      RECORD_BUFFER_LENGTH = 24
❸     CALL GKS$INQ_STROKE_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, DIMENSION, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, TRANSFRM, NUM_POINTS, STROKE_X,
     * STROKE_Y, PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

❹     BUFFER_SIZE = 256
      PROMPT_ECHO_TYPE = 1

C     By specifying to DEC GKS to use the current attributes flag, you
C     need to pass the 24 byte data record instead of the 52 byte record.
      ATTS_FLAG = GKS$K_ACF_CURRENT
```

(continued on next page)

Example 8–9 (Cont.): Using a Stroke Logical Input Device in Request Mode

```
C     Since the device is in request mode by default, initialize the device.
❺     CALL GKS$INIT_STROKE( WS_ID, DEVICE_NUM,
     * NUM_POINTS, STROKE_X, STROKE_Y, TRANSFRM,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )

❻     CALL GKS$SET_STROKE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

❼     CALL GKS$REQUEST_STROKE( WS_ID, DEVICE_NUM,
     * INPUT_STATUS, TRANSFRM, NUM_POINTS, %DESCR( STROKE_X ),
     * %DESCR( STROKE_Y ), RET_SIZE_X, RET_SIZE_Y )

C     Output the input stroke values.
      DO I = 1, RET_SIZE_X, 1
      WRITE(6,*) STROKE_X( I ), STROKE_Y( I )
      END DO
      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The DEC GKS VT241 reads the first six stroke data record components (24 bytes) for prompt and echo type 1. If the sixth component is GKS$K_ACF_CURRENT, then you must specify 24 bytes as the size of the record. If the sixth component is GKS$K_ACF_SPECIFIED, you need to pass the 52-byte record that contains all of the attribute specifications. For more information, see Section 8.2.1, or refer to the *DEC GKS Device Specifics Reference Manual.*

❷ The echo area variable is an array of real numbers representing the rectangular echo area, in device coordinates. The echo area defines a portion of the workstation surface on which the prompt moves.

❸ The function GKS$INQ_STROKE_STATE initializes the variables you need to pass to the input functions. The argument GKS$K_VALUE_REALIZED tells the graphics handler to pass the input values as they are implemented, as opposed to the way that the application may have set the values (GKS$K_VALUE_SET).

After the function call, RECORD_BUFFER_LENGTH contains the amount of the buffer filled with the written data record. If RECORD_SIZE is larger than RECORD_BUFFER_LENGTH, then you know that the data record was truncated to fit into your declared buffer.

❹ This code assigns new values to the input variables. For instance, the buffer size is set to 256. To check the maximum allowable buffer size, call the function GKS$INQ_DEF_STROKE_DATA.

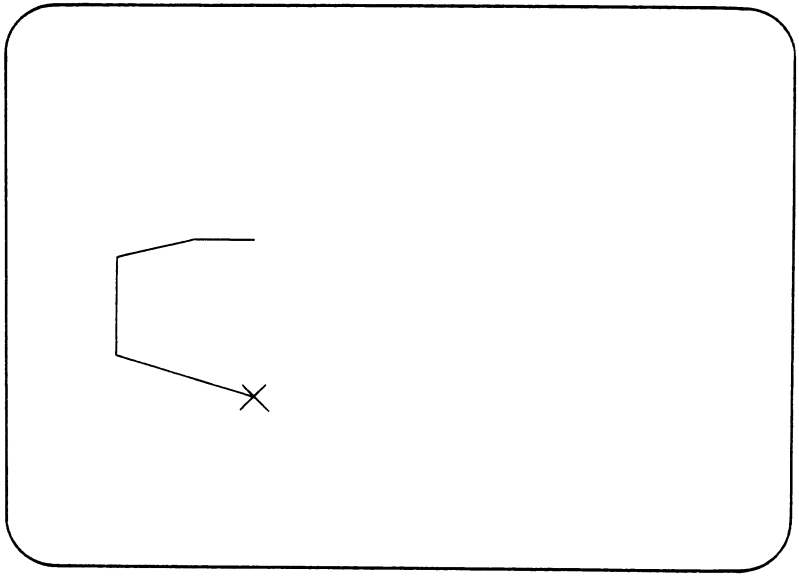
❺ The function GKS$INIT_STROKE initializes the request for stroke input.

❻ The call to GKS$SET_STROKE_MODE places the logical input device into request mode and enables echoing of the input.

❼ The function GKS$REQUEST_STROKE prompts the user for input. The stroke world coordinate values are written to the arguments of this function.

Figure 8–16 shows the screen of a VT241 terminal at the request for input.

Figure 8–16: Requesting from the Stroke Logical Input Device—VT241



INITIALIZE VALUATOR

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$INIT_VALUATOR establishes the initial values of a valuator class logical input device only if the device's prompt is not currently present on the workstation surface (the device must be in request mode). The initial values include the initial valuator value, the prompt and echo type, the echo area, and the data record. Subsequent requests for choice input use the values you specify.

If you do not call GKS$INIT_VALUATOR before you request input from a valuator logical input device, DEC GKS uses the default input values.

Syntax

GKS$INIT_VALUATOR *(workstation_id, device_number, initial_value, echo_type, echo_area, data_record, size_of_record)*

GINVL *(workstation_id, dev_num, lvalue, p_e_type, x_min, x_max, y_min, y_max, low_val, high_val, dim_dr, dr)*

ginitval *(workstation_id,dev, init, pet, area, record)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

initial_value

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument is the real number representing the initial value.

echo_type

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the prompt and echo type.

echo_area

data type: **(real)**
access: **read-only**
mechanism: **by reference**

This argument is the echo area, which is a four-element array specifying the area on the workstation surface on which the prompt appears. Pass the device coordinates in the order X_MINIMUM, X_MAXIMUM, Y_MINIMUM, Y_MAXIMUM.

data_record

data type: **address (record)**
access: **read-only**
mechanism: **by reference**

This argument is a pointer to the data record whose size and contents are dependent on the prompt and echo type, and on the graphics handler requirements. Each workstation may require a different data record structure with different contents.

size_of_record

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the amount of the data record buffer containing the actual data record, in bytes.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-93	DECGKS$_ERROR_NEG_93	Internal GKS error: Prompt and echo type not supported in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
46	GKS$_ERROR_46	Contents of input data record are invalid in routine ****
51	GKS$_ERROR_51	Rectangle definition is invalid in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****

Error Number	Completion Status Code	Message
144	GKS$_ERROR_144	Specified prompt and echo type is not supported on this workstation in routine ****
145	GKS$_ERROR_145	Echo area is outside display space in routine ****
152	GKS$_ERROR_152	Initial value is invalid in routine ****

Program Example

Example 8–10 illustrates the use of the function GKS$INIT_VALUATOR. Following the program example, Figure 8–17 illustrates the program's effect on a VT241 workstation.

Example 8–10: Using a Valuator Logical Input Device in Request Mode

```
C     This program initializes and requests valuator input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH, RECORD_SIZE,
     * INPUT_STATUS, DEVICE_NUM
❶     REAL ECHO_AREA( 4 ), DATA_RECORD( 2 ), UPPER_LIMIT,
     * LOWER_LIMIT, VALUE
❷     DATA WS_ID / 1 /, DEVICE_NUM / 1 /

C     The elements in the data record are the upper and lower limits.
      EQUIVALENCE( DATA_RECORD( 1 ), LOWER_LIMIT )
      EQUIVALENCE( DATA_RECORD( 2 ), UPPER_LIMIT )
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )
```

(continued on next page)

Example 8–10 (Cont.): Using a Valuator Logical Input Device in Request Mode

```
      RECORD_BUFFER_LENGTH = 8
❸     CALL GKS$INQ_VALUATOR_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, VALUE,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

❹     VALUE = 1.5
      UPPER_LIMIT = 3.0
      LOWER_LIMIT = 0.0
      PROMPT_ECHO_TYPE = 1

C     Since the device is in request mode by default, initialize the device.
❺     CALL GKS$INIT_VALUATOR( WS_ID, DEVICE_NUM,
     * VALUE, PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )

❻     CALL GKS$SET_VALUATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

❼     CALL GKS$REQUEST_VALUATOR( WS_ID, DEVICE_NUM,
     * INPUT_STATUS, VALUE )

C     Output the input valuator number.
      WRITE(6,*) VALUE
      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The echo area variable is an array of real numbers representing the rectangular echo area, in device coordinates. The echo area defines a portion of the workstation surface on which the prompt moves.

The DEC GKS VT241 graphics handler uses two components of the valuator input data record for prompt and echo type 1: the real value representing an upper limit, and another real value representing a lower limit.

❷ The VT241 supports three valuator prompt and echo types represented by the integers 1, 2, and 3. Types 1 and 2 prompt the user with a rectangle and a horizontal scale. If using one of these types, the user moves an arrow, using the arrow keys, along the scale between the upper and lower limits. If using type 3, DEC GKS changes a single digital representation of

the real values between the upper and lower limits, the user controlling the change of numbers using the arrow keys.

❸ The function GKS$INQ_VALUATOR_STATE initializes the variables you need to pass to the input functions. After the function call, RECORD_BUFFER_LENGTH contains the amount of the buffer filled with the written data record. If RECORD_SIZE is larger than RECORD_BUFFER_LENGTH, then you know that the data record was truncated to fit into your declared buffer.

❹ This code assigns new values to the input variables. For instance, the upper limit is set to the real value 3.0.

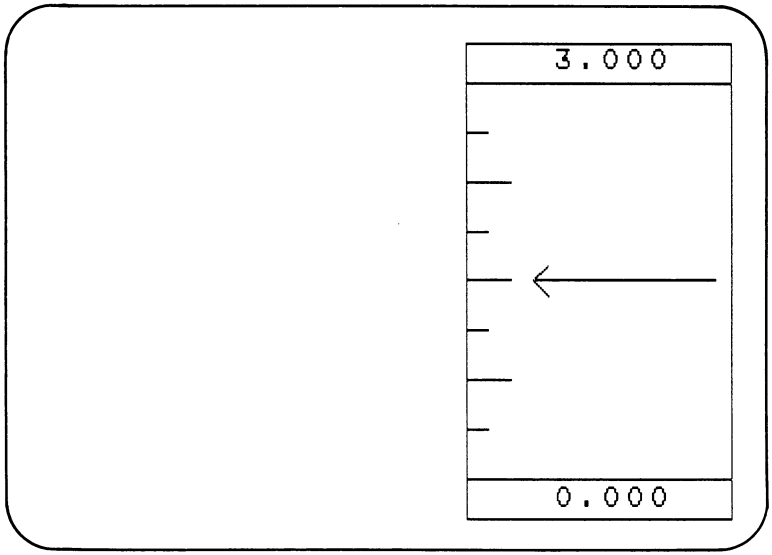
❺ The function GKS$INIT_VALUATOR initializes the request for valuator input.

❻ The call to GKS$SET_VALUATOR_MODE places the logical input device into request mode and enables echoing of the input.

❼ The function GKS$REQUEST_VALUATOR prompts the user for input. The input real value is written to the last argument.

Figure 8–17 shows the screen of a VT241 terminal at the request for input.

Figure 8–17: Requesting from the Valuator Logical Input Device—VT241



ZK-5217-86

Setting Input Operating Modes

This section describes the functions used to control prompt echoing and to change the input operating mode (see Section 8.5 for more information). You do not have to call these functions to initiate the input process. If you choose, you can call the appropriate request function (request is the default input operating mode for DEC GKS) and the logical input device echoes input by default.

If you set the input operating mode to either sample or event mode, the input prompt appears on the workstation surface at the time that you call one of these functions. If you set the input operating mode to request, the prompt does not appear on the workstation surface until you call one of the GKS$REQUEST_ class functions.

This section describes the following functions:

- GKS$SET_MODE_CHOICE
- GKS$SET_MODE_LOCATOR
- GKS$SET_MODE_PICK
- GKS$SET_MODE_STRING
- GKS$SET_MODE_STROKE
- GKS$SET_MODE_VALUATOR

SET CHOICE MODE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_CHOICE_MODE establishes the operating mode of a choice logical input device, and determines whether DEC GKS echoes the prompt and input values on the workstation surface.

Syntax

GKS$SET_CHOICE_MODE *(workstation_id, device_number, operating_mode, echo_flag)*

GSCHM *(workstation_id, dev_num, operating_mode, echo)*

gsetchoicemode *(workstation_id, dev, operating_mode, echo)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

operating_mode

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the operating mode specifying the method of input. This argument can be any of the following values, according to the DEC GKS standard:

Value	Constant	Description
0	GKS$K_INPUT_MODE_REQUEST	Request mode
1	GKS$K_INPUT_MODE_SAMPLE	Sample mode
2	GKS$K_INPUT_MODE_EVENT	Event mode

echo_flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the echo flag. This flag determines whether or not the prompt and input values are echoed on the workstation surface. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_NOECHO	Disable the echo.
1	GKS$K_ECHO	Enable the echo.

Error Messages

Error Number	Completion Status Code	Message
-16	DECGKS$_ERROR_NEG_16	Echo switch is invalid in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–6.

SET LOCATOR MODE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_LOCATOR_MODE establishes the operating mode of a locator logical input device, and determines whether DEC GKS echoes the prompt and input values on the workstation surface.

Syntax

GKS$SET_LOCATOR_MODE *(workstation_id, device_number, operating_mode, echo_flag)*

GSLCM *(workstation_id, dev_num, operating_mode, echo)*

gsetlocmode *(workstation_id, dev,operating_mode, echo)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

operating_mode

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the operating mode specifying the method of input. This argument can be any of the following values, according to the ANSI GKS standard:

Value	Constant	Description
0	GKS$K_INPUT_MODE_REQUEST	Request mode
1	GKS$K_INPUT_MODE_SAMPLE	Sample mode
2	GKS$K_INPUT_MODE_EVENT	Event mode

echo_flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the echo flag. This flag determines whether or not the prompt and input values are echoed on the workstation surface. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_NOECHO	Disable the echo.
1	GKS$K_ECHO	Enable the echo.

Error Messages

Error Number	Completion Status Code	Message
-16	DECGKS$_ERROR_NEG_16	Echo switch is invalid in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–1.

SET PICK MODE

Operating States: WSOP, WSAC, SGOP

Description

This function establishes the operating mode of a pick logical input device, and determines whether DEC GKS echoes the prompt and input values on the workstation surface.

Syntax

GKS$SET_PICK_MODE *(workstation_id, device_number, operating_mode, echo_flag)*

GSPKM *(workstation_id, dev_num, operating_mode, echo)*

gsetpickmode *(workstation_id, dev,operating_mode, echo)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation. To see if your workstation supports more than one logical device type, refer to the *DEC GKS Device Specifics Reference Manual.*

operating_mode

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the operating mode specifying the method of input. This argument could be any of the following values, according to the ANSI GKS standard:

Value	Constant	Description
0	GKS$K_INPUT_MODE_REQUEST	Request mode
1	GKS$K_INPUT_MODE_SAMPLE	Sample mode
2	GKS$K_INPUT_MODE_EVENT	Event mode

echo_flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the echo flag. This flag determines whether or not the prompt and input values are echoed on the workstation surface. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_NOECHO	Disable the echo.
1	GKS$K_ECHO	Enable the echo.

Error Messages

Error Number	Completion Status Code	Message
-16	DECGKS$_ERROR_NEG_16	Echo switch is invalid in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–7.

SET STRING MODE

Operating States: WSOP, WSAC, SGOP

Description

This function establishes the operating mode of a string logical input device, and specifies whether DEC GKS echoes the prompt and input values on the workstation surface.

Syntax

GKS$SET_STRING_MODE *(workstation_id, device_number, operating_mode, echo_flag)*

GSSTM *(workstation_id, dev_num, operating_mode, echo)*

gsetstringmode *(workstation_id, dev,operating_mode, echo)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

operating_mode

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the operating mode specifying the method of input. This argument could be any of the following values, according to the ANSI GKS standard:

Value	Constant	Description
0	GKS$K_INPUT_MODE_REQUEST	Request mode
1	GKS$K_INPUT_MODE_SAMPLE	Sample mode
2	GKS$K_INPUT_MODE_EVENT	Event mode

echo_flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the echo flag. This flag determines whether or not the prompt and input values are echoed on the workstation surface. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_NOECHO	Disable the echo.
1	GKS$K_ECHO	Enable the echo.

Error Messages

Error Number	Completion Status Code	Message
-16	DECGKS$_ERROR_NEG_16	Echo switch is invalid in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–8.

SET STROKE MODE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_STROKE_MODE establishes the operating mode of a stroke logical input device, and determines whether DEC GKS echoes the prompt and input values on the workstation surface.

Syntax

GKS$SET_STROKE_MODE *(workstation_id, device_number, operating_mode, echo_flag)*

GSSKM *(workstation_id, dev_num, operating_mode, echo)*

gsetstringmode *(workstation_id, dev,operating_mode, echo)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

operating_mode

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the operating mode specifying the method of input. This argument could be any of the following values, according to the ANSI GKS standard:

Value	Constant	Description
0	GKS$K_INPUT_MODE_REQUEST	Request mode
1	GKS$K_INPUT_MODE_SAMPLE	Sample mode
2	GKS$K_INPUT_MODE_EVENT	Event mode

echo_flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the echo flag. This flag determines whether or not the prompt and input values are echoed on the workstation surface. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_NOECHO	Disable the echo.
1	GKS$K_ECHO	Enable the echo.

Error Messages

Error Number	Completion Status Code	Message
-16	DECGKS$_ERROR_NEG_16	Echo switch is invalid in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–9.

SET VALUATOR MODE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_VALUATOR_MODE establishes the operating mode of a valuator logical input device, and determines whether DEC GKS echoes the prompt and input values on the workstation surface.

Syntax

GKS$SET_VALUATOR_MODE *(workstation_id, device_number, operating_mode, echo_flag)*

GSVLM *(workstation_id, dev_num, operating_mode, echo)*

gsetvalmode *(workstation_id, dev,operating_mode, echo)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

operating_mode

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the operating mode specifying the method of input. This argument could be any of the following values, according to the ANSI GKS standard:

Value	Constant	Description
0	GKS$K_INPUT_MODE_REQUEST	Request mode
1	GKS$K_INPUT_MODE_SAMPLE	Sample mode
2	GKS$K_INPUT_MODE_EVENT	Event mode

echo_flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the echo flag. This flag determines whether or not the prompt and input values are echoed on the workstation surface. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_NOECHO	Disable the echo.
1	GKS$K_ECHO	Enable the echo.

Error Messages

Error Number	Completion Status Code	Message
-16	DECGKS$_ERROR_NEG_16	Echo switch is invalid in routine ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–10.

Requesting Input

This section describes the functions used to initiate the request for input from a logical input device. With DEC GKS, the default input operating mode is request mode. See Section 8.5 for information concerning the different types of input operating modes.

This section describes the following functions:

- GKS$REQUEST_CHOICE
- GKS$REQUEST_LOCATOR
- GKS$REQUEST_PICK
- GKS$REQUEST_STRING
- GKS$REQUEST_STROKE
- GKS$REQUEST_VALUATOR

REQUEST CHOICE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$REQUEST_CHOICE prompts the user for input according to the specifications you may have passed to GKS$INIT_CHOICE and GKS$SET_CHOICE_MODE. At this point in the application program, the user makes a selection from several possibilities (for example, by moving the cursor through a menu) and then signals whether or not the input is valid.

If the user accepts the input, the function writes GKS$K_STATUS_OK to the status argument, and the positive integer representing the user's choice to the input argument.

If the user invokes a break action, the function returns GKS$K_STATUS_NONE to the status argument, and the value 0 to the input argument. For choice class logical input devices, the value 0 indicates a break; the status GKS$K_STATUS_OK indicates input; and the status GKS$K_STATUS_NOCHOICE indicates that the user did not make a choice (input was triggered without the cursor being moved).

Syntax

GKS$REQUEST_CHOICE *(workstation_id, device_number, input_status, choice_value)*

GRQCH *(workstation_id, dev_num, in_status, ch_num)*

greqchoice *(workstation_id, dev, response)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

input_status

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the status of the input process. This argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_STATUS_NONE	Input break.
1	GKS$K_STATUS_OK	Input obtained.
2	GKS$K_STATUS_NOCHOICE	Triggered without choosing.

choice_value

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer representing the user's choice.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–6.

REQUEST LOCATOR

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$REQUEST_LOCATOR prompts the user for input according to the specifications you may have passed to GKS$INIT_LOCATOR and GKS$SET_LOCATOR_MODE. At this point in the application program, the user positions the cursor within the echo area, indicating a device coordinate corresponding to a world coordinate point, and then signals whether or not the input is valid.

For more information about the locator position and echo types 2 and 3, see the Chapter 1, VAXstation Workstation Specifics, in the *DEC GKS Device Specifics Reference Manual.*

If the user accepts the input, the function writes GKS$K_STATUS_OK to the status argument and writes the information about the input point to the last three arguments. One argument contains the transformation number used to transform the device coordinate to a world coordinate point. The remaining arguments contain the corresponding world coordinate points.

If the user invokes a break action, the function writes GKS$K_STATUS_NONE to the status argument. DEC GKS ignores the current input values of the locator class device if the user invokes a break action.

Syntax

GKS$REQUEST_LOCATOR *(workstation_id, device_number, input_status, transformation_number, world_x, world_y)*

GRQLC *(workstation_id, dev_num, in_status, x_form, pos_x, pos_y)*

greqloc *(workstation_id, dev, response)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

input_status

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the status of the input process. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_STATUS_NONE	No input obtained.
1	GKS$K_STATUS_OK	Input obtained.

transformation_number

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the number of the normalization transformation used to translate the input point to a world coordinate point.

world_x
world_y

data type: **real**
access: **write-only**
mechanism: **by reference**

These are the arguments to which DEC GKS writes the X and Y world coordinate values.

Error Messages

Error Number	Completion Status Code	Message
–20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–1.

REQUEST PICK

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$REQUEST_PICK prompts the user for input according to the specifications you may have passed to GKS$INIT_PICK and GKS$SET_PICK_MODE. At this point in the application program, the user positions a cursor on a portion of a segment and then signals whether or not the input is valid.

If the user accepts the input, the function writes GKS$K_STATUS_OK to the status argument, and writes the integers representing the name of the chosen segment and the chosen pick identifier (refer to GKS$SET_PICK_ID in Chapter 9, Segment Functions) to the last two arguments.

If the user invokes a break action, the function returns GKS$K_STATUS_NONE to the status argument, and the input values are invalid. If the user triggered the input measure before moving the prompt, or if the user triggers input while the cursor is not positioned on a segment, then this function writes GKS$K_STATUS_NOPICK to the status argument.

Syntax

GKS$REQUEST_PICK *(workstation_id, device_number, input_status, segment_name, pick_id)*

GRQPK *(workstation_id, dev_num, in_status, segment_name, pick_id)*

greqpick *(workstation_id, dev, response)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

input_status

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the status of the input process. This argument can be any of the following values or constants:

Value	Constant	Description
0	GKS$K_STATUS_NONE	Break during input.
1	GKS$K_STATUS_OK	Input obtained.
2	GKS$K_STATUS_NOPICK	Input triggered without picking.

segment_name

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer representing the chosen segment.

pick_id

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer pick identifier value associated with the picked primitive within the segment.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
37	GKS$_ERROR_37	Specified workstation is not of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****

Program Example

For an example of a call to this function, refer to Example 8–7.

REQUEST STRING

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$REQUEST_STRING prompts the user for input according to the specifications you may have passed to GKS$INIT_STRING and GKS$SET_STRING_MODE. At this point in the application program, the user enters a string of characters at the prompt and then signals whether or not the input is valid.

When requesting string input, the following two buffers exist:

- The application's string buffer, whose size you specify when you pass the buffer argument by descriptor to GKS$REQUEST_STRING.
- The logical input device's string buffer, whose size you can specify in the call to GKS$INIT_STRING.

If the user accepts the input, the function writes GKS$K_STATUS_OK to the status argument, the character string to the application's buffer, and the length of the character string to the last argument. If the entered string is larger than the application's buffer, then you lose all additional data. You must make sure that your application's buffer is as large as the device's string buffer.

If the user invokes a break action, the function returns GKS$K_STATUS_NONE to the status argument, and the input arguments are not valid.

Syntax

GKS$REQUEST_STRING *(workstation_id, device_number, input_status, string_buffer, string_size)*

GRQST *(workstation_id, dev_num, in_status, num_char, cstring)*

GRQST - Subset *(workstation_id, dev_num, in_status, num_char, cstring)*

greqstring *(workstation_id, dev, response)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation. To see if your workstation supports more than one logical device type, refer to the *DEC GKS Device Specifics Reference Manual.*

input_status

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the status of the input process. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_STATUS_NONE	No input obtained.
1	GKS$K_STATUS_OK	Input obtained.

string_buffer

data type: **string**
access: **write-only**
mechanism: **by descriptor, data type in descriptor**

This is the argument to which DEC GKS writes the input character string. This is the application's string buffer.

string_size

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the size of the character string, in bytes.

Error Messages

Error Number	Completion Status Code	Message
–20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–8.

REQUEST STROKE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$REQUEST_STROKE prompts the user for input according to the specifications you may have passed to GKS$INIT_STROKE and GKS$SET_STROKE_MODE. At this point in the application program, the user designates certain points to be contained in the stroke and then signals whether or not the input is valid.

If the user accepts the input, the function writes GKS$K_STATUS_OK to the status argument, and writes the normalization transformation number used to translate the device coordinates to world coordinate points, the returned stroke points, the number of entered points, and the number of accepted points to the corresponding output arguments.

When requesting stroke input, the following two buffers exist:

- The application's stroke buffer, whose size you specify when you pass the buffer argument by descriptor to GKS$REQUEST_STROKE.
- The logical input device's stroke buffer, whose size you can specify in the call to GKS$INIT_STROKE.

DEC GKS can return points up to the number specified by the size of the application's X and Y coordinate buffers. If the size of the entered stroke is larger than the number of points placed in the application's buffers, you lose all additional data. You must make sure that your application's buffers are as large as the device's stroke buffers.

If the user invokes a break action, the function returns GKS$K_STATUS_NONE to the status argument, and the input values are not valid.

Syntax

GKS$REQUEST_STROKE *(workstation_id, device_number, input_status, transformation_number, num_entered_points, stroke_buffer_x, stroke_buffer_y, stroke_size_x, stroke_size_y)*

GRQSK *(workstation_id, dev_num, max_pts, in_status, xform, num_pts, px, py)*

greqstroke *(workstation_id, dev, response)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

input_status

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This is the argument to which DEC GKS writes the status of the input process. This argument can be either of the following values or constants.

Value	Constant	Description
0	GKS$K_STATUS_NONE	No input obtained.
1	GKS$K_STATUS_OK	Input obtained.

transformation_number

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the number of the normalization transformation used to translate the input points to world coordinate points.

num_entered_points

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the number of points in the stroke entered by the user.

stroke_buffer_x
stroke_buffer_y

data type: **array (real)**
access: **write-only**
mechanism: **by reference**

These are the arguments to which DEC GKS writes the X and Y world coordinate values of the accepted stroke. These arguments are the application's stroke buffer.

stroke_size_x
stroke_size_y

data type: **integer**
access: **write-only**
mechanism: **by reference**

These are the arguments to which DEC GKS writes the number of stroke points that DEC GKS actually accepted.

Error Messages

Error Number	Completion Status Code	Message
–20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****

Program Example

For an example of a call to this function, refer to Example 8–9.

REQUEST VALUATOR

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$REQUEST_VALUATOR prompts the user for input according to the specifications you may have passed to GKS$INIT_VALUATOR and GKS$SET_VALUATOR_MODE. At this point in the application program, the user selects a value within a defined range and then signals whether or not the input is valid.

If the user accepts the input, the function writes GKS$K_STATUS_OK to the status argument, and the selected real number to the input argument.

If the user invokes a break action, the function returns GKS$K_STATUS_NONE to the status argument, and the input value is not valid.

Syntax

GKS$REQUEST_VALUATOR *(workstation_id, device_number, input_status, real_value)*

GRQVL *(workstation_id, dev_num, in_status, value)*

greqval *(workstation_id, dev, response)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

input_status

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the status of the input process. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_STATUS_NONE	No input obtained.
1	GKS$K_STATUS_OK	Input obtained.

real_value

data type: **real**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the real number chosen by the user.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
141	GKS$_ERROR_141	Input device is not in REQUEST mode in routine ****

Program Example

For an example of a call to this function, refer to Example 8–10.

Sampling Input

This section describes the functions used to sample the current measure of a logical input device. DEC GKS returns the measure of the device without requiring a trigger from the user. See Section 8.5 for information concerning the different types of input operating modes.

This section describes the following functions:

- GKS$SAMPLE_CHOICE
- GKS$SAMPLE_LOCATOR
- GKS$SAMPLE_PICK
- GKS$SAMPLE_STRING
- GKS$SAMPLE_STROKE
- GKS$SAMPLE_VALUATOR

SAMPLE CHOICE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SAMPLE_CHOICE writes the current measure of the specified choice logical input device to the corresponding output argument.

If the input is valid, the function writes GKS$K_STATUS_OK to the status argument and writes the positive integer representing the user's choice to the input argument.

If the initial choice status is GKS$K_STATUS_NOCHOICE, and if the user did not move the prompt from its initial position, this function writes GKS$K_STATUS_NOCHOICE to the status argument (this indicates that the user did not make a choice yet).

Syntax

GKS$SAMPLE_CHOICE *(workstation_id, device_number, input_status, choice_value)*

GSMCH *(workstation_id, dev_num, in_status, ch_num)*

gsamplechoice *(workstation_id, dev, response)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

input_status

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the status of the input process. This argument can be either of the following values or constants:

Value	Constant	Description
1	GKS$K_STATUS_OK	Input obtained.
2	GKS$K_STATUS_NOCHOICE	Sampled without choosing.

choice_value

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer representing the user's choice.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
142	GKS$_ERROR_142	Input device is not in SAMPLE mode in routine ****

Program Example

Example 8–11 illustrates the use of the function GKS$SAMPLE_CHOICE. Following the program example, Figures 8–18 through 8–20 illustrate the program's effect on a VT241 workstation.

Example 8–11: Using a Choice Logical Input Device in Sample Mode

```
C     This program initializes and Samples choice input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, DATA_RECORD( 3 ), NUM_CHOICES, SIZES( 3 ),
     * ADDRESSES( 3 ), PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH, RECORD_SIZE,
     * INPUT_STATUS, INITIAL_CHOICE, DEVICE_NUM, INPUT_CHOICE,
     * INITIAL_STATUS, NUM_POINTS, COLOR1, COLOR2, COLOR3
      REAL ECHO_AREA( 4 ), PX( 4 ), PY( 4 ), LARGER
      DATA PX / 0.05, 0.1, 0.075, 0.05 /
      DATA PY / 0.85, 0.85, 0.80, 0.85 /
      DATA NUM_POINTS / 4 /, COLOR1 / 1 /, COLOR2 / 2 /,
     * COLOR3 / 3 /, LARGER / 0.03 /

      CHARACTER*80 CURRENT_STRINGS( 3 )

      DATA WS_ID / 1 /, DEVICE_NUM / 1 /

C     First element in the data record is the number of choices.
      EQUIVALENCE( DATA_RECORD(1), NUM_CHOICES )
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Establish the size of the record buffer: 12 bytes.
      RECORD_BUFFER_LENGTH = 12

C     The second element in the VT241 choice data record is the pointer to
C     the array containing sizes of each choice character string. You need
C     to initialize the pointer so that the array can be initialized.
      DATA_RECORD( 2 ) = %LOC( SIZES(1) )

C     The third element in the VT241 choice data record is the pointer to the
C     array containing the pointers to the strings to be used. You need
C     to initialize the pointer so that the array can be initialized.
      DATA_RECORD( 3 ) = %LOC( ADDRESSES(1) )
      ADDRESSES( 1 ) = %LOC( CURRENT_STRINGS( 1 ) )
      ADDRESSES( 2 ) = %LOC( CURRENT_STRINGS( 2 ) )
      ADDRESSES( 3 ) = %LOC( CURRENT_STRINGS( 3 ) )

C     Inquire about the current values.
      NUM_CHOICES = 3
      CALL GKS$INQ_CHOICE_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, INITIAL_STATUS,
     * INITIAL_CHOICE, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH, RECORD_SIZE )
```

(continued on next page)

Example 8–11 (Cont.): Using a Choice Logical Input Device in Sample Mode

```
C     Set the initial choice status.
      INITIAL_STATUS = GKS$K_STATUS_OK

C     Establish sizes of prompt strings...
      SIZES( 1 ) = 4
      SIZES( 2 ) = 5
      SIZES( 3 ) = 4

C     Establish locations of prompt strings...
      ADDRESSES( 1 ) = %LOC( 'Pink' )
      ADDRESSES( 2 ) = %LOC( 'Green' )
      ADDRESSES( 3 ) = %LOC( 'Blue' )

C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_CHOICE( WS_ID, DEVICE_NUM,
     * INITIAL_STATUS, INITIAL_CHOICE, PROMPT_ECHO_TYPE,
     * ECHO_AREA, DATA_RECORD, RECORD_BUFFER_LENGTH )

❶     CALL GKS$SET_CHOICE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_SAMPLE, GKS$K_ECHO )

C     Initialize color indexes.
      CALL GKS$SET_COLOR_REP( WS_ID, COLOR1, 0.6258, 0.2142,
     * 0.2142 )
      CALL GKS$SET_COLOR_REP( WS_ID, COLOR2, 0.1400, 1.000,
     * 0.1400 )
      CALL GKS$SET_COLOR_REP( WS_ID, COLOR3, 0.0, 0.0,
     * 0.8400 )
      CALL GKS$SET_FILL_COLOR_INDEX( COLOR1 )
      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )
```

(continued on next page)

Example 8–11 (Cont.): Using a Choice Logical Input Device in Sample Mode

```
C     Tell the user how to change colors.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.05, 0.95, 'Move the arrow keys to')
      CALL GKS$TEXT( 0.05, 0.90, 'change the triangle colors.')

C     Do until the surface is full.
      DO WHILE ( PX( 2 ) .LT. 0.95 )

❷     CALL GKS$SAMPLE_CHOICE( WS_ID, DEVICE_NUM,
     * INPUT_STATUS, INPUT_CHOICE )

C     Depending on the sample, change the color.
      IF ( INPUT_CHOICE .EQ. 1 ) THEN
         CALL GKS$SET_FILL_COLOR_INDEX( COLOR1 )
      ENDIF
      IF ( INPUT_CHOICE .EQ. 2 ) THEN
         CALL GKS$SET_FILL_COLOR_INDEX( COLOR2 )
      ENDIF
      IF ( INPUT_CHOICE .EQ. 3 ) THEN
         CALL GKS$SET_FILL_COLOR_INDEX( COLOR3 )
      ENDIF

❸     CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )

      PY( 1 ) = PY( 1 ) - 0.06
      PY( 2 ) = PY( 2 ) - 0.06
      PY( 3 ) = PY( 3 ) - 0.06
      PY( 4 ) = PY( 4 ) - 0.06

      IF ( PY( 2 ) .LT. 0.05 ) THEN
         PY( 1 ) = 0.85
         PY( 2 ) = 0.85
         PY( 3 ) = 0.80
         PY( 4 ) = 0.85
         PX( 1 ) = PX( 1 ) + 0.06
         PX( 2 ) = PX( 2 ) + 0.06
         PX( 3 ) = PX( 3 ) + 0.06
         PX( 4 ) = PX( 4 ) + 0.06
      ENDIF

      ENDDO
```

(continued on next page)

Example 8–11 (Cont.): Using a Choice Logical Input Device in Sample Mode

```
C     Turn off the sample prompt.
❹     CALL GKS$SET_CHOICE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The call to GKS$SET_CHOICE_MODE sets the input operating mode to sample. At this point in the program, the choice prompt appears on the workstation surface and the user can change the measure of the device.

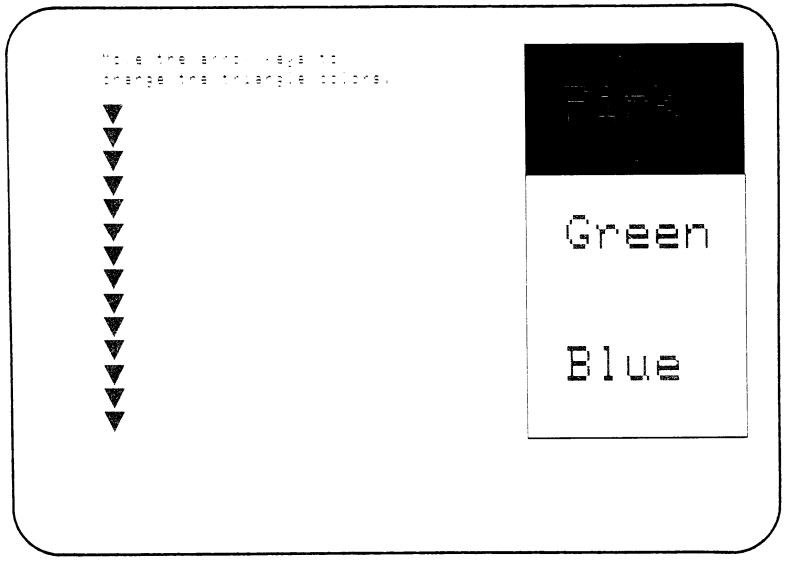
❷ This call to GKS$SAMPLE_CHOICE retrieves the current input value (without the user having to trigger the device). The WHILE loop ends when the program fills the workstation surface with triangles.

❸ This code draws triangles in columns.

❹ The call to GKS$SET_CHOICE_MODE returns the logical input device to request mode. At this point, the device handler removes the choice prompt from the workstation surface and the user can no longer enter input.

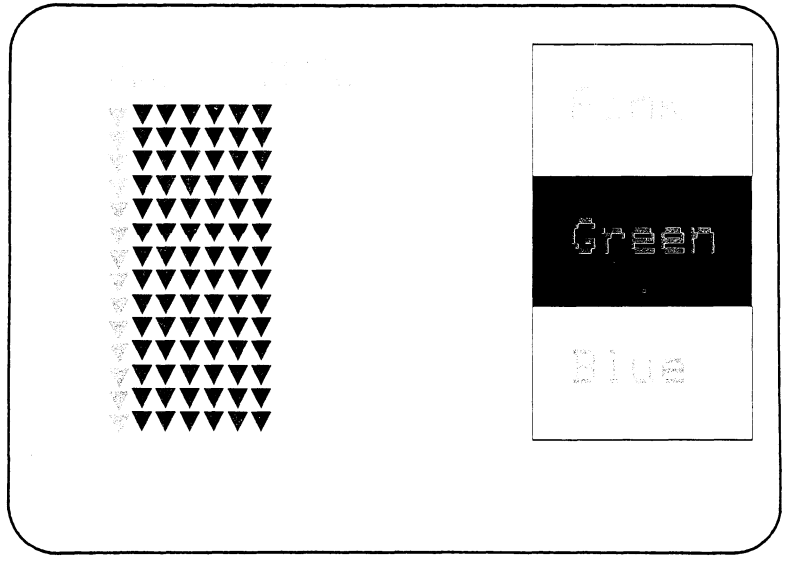
Figure 8–18 illustrates the surface of the VT241 when the input mode is set. Figure 8–19 illustrates the surface of the VT241 when the user moves the prompt to the second choice. Notice that the user need only move the prompt to another color (without triggering) and the color of the triangles change accordingly. Figure 8–20 illustrates the surface of the VT241 when the workstation surface is full.

Figure 8–18: The Choice Logical Input Device in Sample Mode—VT241



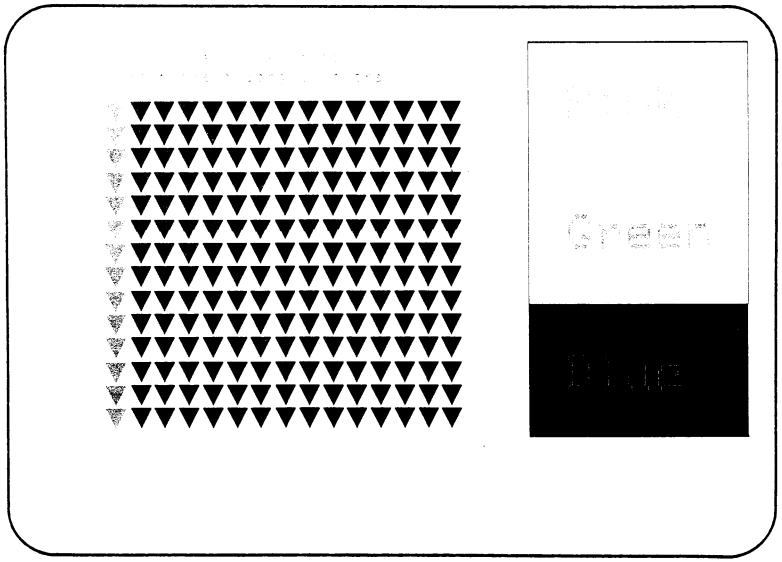
ZK-5822-HC

Figure 8–19: The Choice Logical Input Device in Sample Mode—VT241



ZK-5824-HC

Figure 8–20: The Choice Logical Input Device in Sample Mode—VT241



ZK-5825-HC

SAMPLE LOCATOR

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SAMPLE_LOCATOR writes the current measure of the specified device and the corresponding normalization transformation number to the appropriate output arguments.

Syntax

GKS$SAMPLE_LOCATOR *(workstation_id, device_number, transformation_number, world_x, world_y)*

GSMLC *(workstation_id, dev_num, x_form, pos_x, pos_y)*

gsampleloc *(workstation_id, dev, response)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

transformation_number

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the normalization transformation number used to translate the input point to a world coordinate point.

world_x
world_y

data type: **real**
access: **write-only**
mechanism: **by reference**

These are the arguments to which DEC GKS writes the X and Y world coordinate values.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****

Error Number	Completion Status Code	Message
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
142	GKS$_ERROR_142	Input device is not in SAMPLE mode in routine ****

Program Example

For an example of a call to this function, refer to Example 8–2.

SAMPLE PICK

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SAMPLE_PICK writes the current measure of the specified pick logical input device to the corresponding output argument.

If the input is valid, the function writes GKS$K_STATUS_OK to the status argument and writes the positive integers representing the picked segment and the pick identifier to the output arguments.

If the initial choice status is GKS$K_STATUS_NOPICK, and if the user did not move the prompt, this function writes GKS$K_STATUS_NOPICK to the status argument (this indicates that the user did not pick a segment yet). The logical input device also returns GKS$K_STATUS_NOPICK if the user moved the prompt but the aperture is not touching a segment at the time of the sample.

Syntax

GKS$SAMPLE_PICK *(workstation_id, device_number, input_status, segment_name, pick_id)*

GSMPK *(workstation_id, dev_num, in_status, segment_name, pick_id)*

gsamplepick *(workstation_id, dev, response)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

input_status

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the status of the input process. This argument can be either of the following values or constants:

Value	Constant	Description
1	GKS$K_STATUS_OK	Input obtained.
2	GKS$K_STATUS_NOPICK	Sampled without picking.

segment_name

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer representing the chosen segment.

pick_id

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer pick identifier value associated with the picked primitive within the segment. For more information, refer to Chapter 9, Segment Functions.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
37	GKS$_ERROR_37	Specified workstation is not of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
142	GKS$_ERROR_142	Input device is not in SAMPLE mode in routine ****

Program Example

Example 8–12 illustrates the use of the function GKS$SAMPLE_PICK. Following the program example, Figures 8–21 through 8–23 illustrate the program's effect on a VT241 workstation.

Example 8–12: Using a Pick Logical Input Device in Sample Mode

```
C     This program initializes and samples pick input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, INITIAL_STATUS, SEGMENT,
     * PICK_ID, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE, INPUT_STATUS,
     * DEVICE_NUM, BOX_1, BOX_2, TRIANGLE_1,
     * TRIANGLE_2, NUM_POINTS
      REAL ECHO_AREA(4), DATA_RECORD( 1 )
      REAL X_VALUES( 4 ), Y_VALUES( 4 ), LARGER
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, BOX_1 / 1 /,
     * BOX_2 / 2 /, TRIANGLE_1 / 1 /, TRIANGLE_2 / 2 /,
     * NUM_POINTS / 4 /, LARGER / 0.03 /
      DATA X_VALUES / 0.1, 0.4, 0.1, 0.1 /
      DATA Y_VALUES / 0.3, 0.6, 0.6, 0.3 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

(1)   CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )

      CALL GKS$CREATE_SEG( BOX_1 )
      CALL GKS$SET_PICK_ID( TRIANGLE_1 )
      CALL GKS$SET_FILL_COLOR_INDEX( 2 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      X_VALUES( 3 ) = 0.4
      Y_VALUES( 3 ) = 0.3
      CALL GKS$SET_PICK_ID( TRIANGLE_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 3 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      CALL GKS$CLOSE_SEG()

      X_VALUES( 1 ) = 0.6
      X_VALUES( 2 ) = 0.9
      X_VALUES( 3 ) = 0.6
      X_VALUES( 4 ) = 0.6
      Y_VALUES( 3 ) = 0.6

      CALL GKS$SET_PICK_ID( TRIANGLE_1 )
```

(continued on next page)

Example 8–12 (Cont.): Using a Pick Logical Input Device in Sample Mode

```
      CALL GKS$CREATE_SEG( BOX_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 2 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      X_VALUES( 3 ) = 0.9
      Y_VALUES( 3 ) = 0.3
      CALL GKS$SET_PICK_ID( TRIANGLE_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 3 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      CALL GKS$CLOSE_SEG()

      CALL GKS$SET_SEG_DETECTABILITY( BOX_1, GKS$K_DETECTABLE )
      CALL GKS$SET_SEG_DETECTABILITY( BOX_2, GKS$K_DETECTABLE )

      CALL GKS$SET_TEXT_HEIGHT( 0.03 )
      CALL GKS$TEXT( 0.2, 0.45, '1')
      CALL GKS$TEXT( 0.3, 0.45, '2')
      CALL GKS$TEXT( 0.7, 0.45, '1')
      CALL GKS$TEXT( 0.8, 0.45, '2')

C     Declare a data length of one long word which will hold the
C     size of the pick prompt.
      RECORD_BUFFER_LENGTH = 4
      CALL GKS$INQ_PICK_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE,
     * ECHO_FLAG, INITIAL_STATUS, SEGMENT, PICK_ID,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

C     Establish initial values.
      SEGMENT = BOX_1
      PICK_ID = TRIANGLE_1
      INITIAL_STATUS = GKS$K_STATUS_OK

C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_PICK( WS_ID, DEVICE_NUM, INITIAL_STATUS,
     * SEGMENT, PICK_ID, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH )
```

(continued on next page)

Example 8–12 (Cont.): Using a Pick Logical Input Device in Sample Mode

```
❷       CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
       * GKS$K_INPUT_MODE_SAMPLE, GKS$K_ECHO )

C      Tell the user the task.
       CALL GKS$SET_TEXT_HEIGHT( LARGER )
       CALL GKS$TEXT( 0.05, 0.95, 'Move the cursor to a triangle.' )
       CALL GKS$TEXT( 0.05, 0.90, 'I will say if it is correct.')

C      Do until the user picks the second triangle in the second box.
       DO WHILE (( SEGMENT .NE. 2 ) .OR. ( PICK_ID .NE. 2 ))

❸       CALL GKS$SAMPLE_PICK( WS_ID, DEVICE_NUM, INPUT_STATUS,
       * SEGMENT, PICK_ID )

C      Tease the user as s/he gets closer.
❹       IF (( SEGMENT .EQ. 1 ) .AND. ( PICK_ID .EQ. 1 )) THEN
          CALL GKS$TEXT( 0.05, 0.85,
       *                 'You are pretty far away.')
        ENDIF
        IF (( SEGMENT .EQ. 1 ) .AND. ( PICK_ID .EQ. 2 )) THEN
          CALL GKS$TEXT( 0.05, 0.80,
       *                 'You are getting closer.')
        ENDIF
        IF (( SEGMENT .EQ. 2 ) .AND. ( PICK_ID .EQ. 1 )) THEN
          CALL GKS$TEXT( 0.05, 0.75,
       *                 'You are REALLY close.')
        ENDIF
        ENDDO

        CALL GKS$TEXT( 0.05, 0.70, 'YOU MADE IT!!!!')

C      Turn off the sample prompt.
❺       CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
       * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

        CALL GKS$DEACTIVATE_WS( WS_ID )
        CALL GKS$CLOSE_WS( WS_ID )
        CALL GKS$CLOSE_GKS()
        END
```

The following numbers correspond to the numbers in the previous example:

❶ This code establishes the same divided boxes that appear in Example 8–7.

❷ The call to GKS$SET_PICK_MODE sets the input operating mode to sample. At this point in the program, the pick aperture appears on the workstation surface and the user can change the measure of the device.

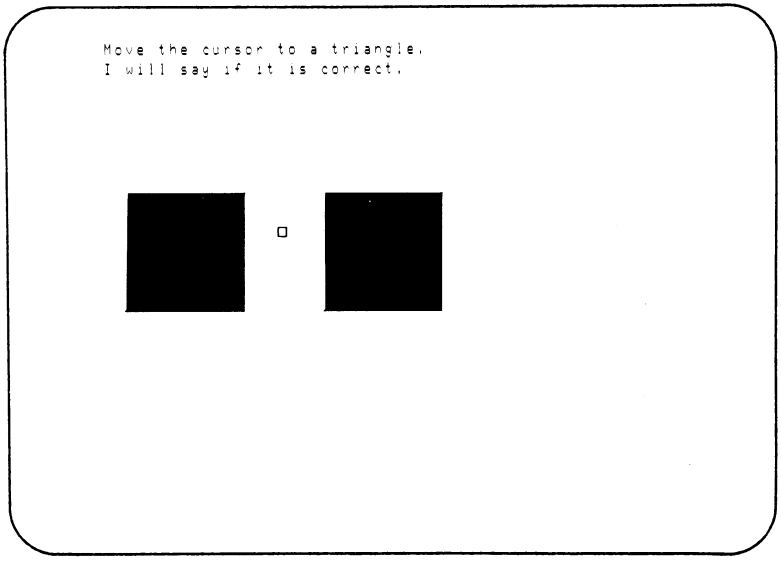
❸ This call to GKS$SAMPLE_PICK retrieves the current input value (without the user having to trigger the device). The WHILE loop ends when the user picks the second triangle in the second box.

❹ This code teases the user, saying how close the aperture is to segment 2, pick identifier 2.

❺ The call to GKS$SET_PICK_MODE returns the logical input device to request mode. At this point, the device handler removes the aperture from the workstation surface and the user can no longer enter input.

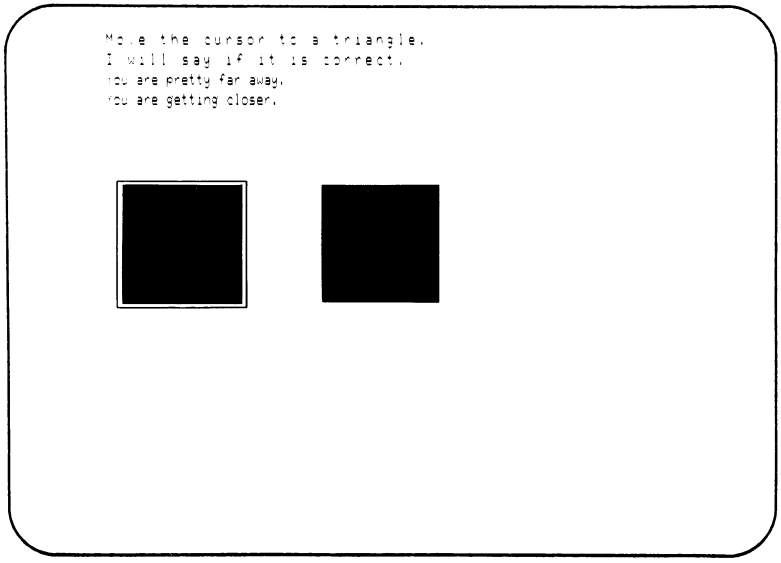
Figure 8–21 illustrates the surface of the VT241 when the input mode is set. Figure 8–22 illustrates the surface of the VT241 when the user moves the aperture closer to the required segment and pick identifier. Notice that the user need only move the aperture to another pick identifier (without triggering) and a new message appears on the workstation surface. Figure 8–23 illustrates the surface of the VT241 when the user picks the correct triangle.

Figure 8–21: The Pick Logical Input Device in Sample Mode—VT241



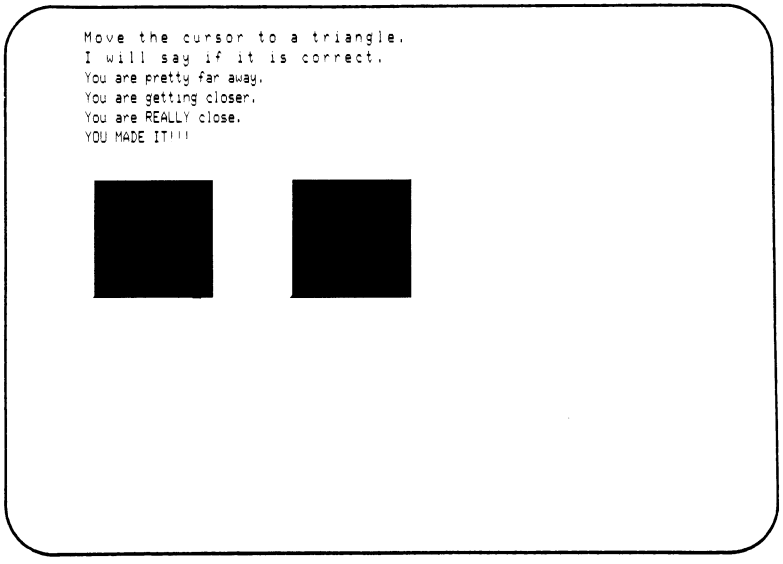
ZK-5838-HC

Figure 8–22: The Pick Logical Input Device in Sample Mode—VT241



ZK-5820-HC

Figure 8–23: The Pick Logical Input Device in Sample Mode—VT241



ZK-5821-HC

SAMPLE STRING

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SAMPLE_STRING writes the current measure of the specified string logical input device to the appropriate output arguments.

When activating string input, the following two buffers exist:

- The application's string buffer, whose size you specify when you pass the buffer argument by descriptor to GKS$SAMPLE_STRING.
- The logical input device's string buffer, whose size you can specify in the call to GKS$INIT_STRING.

When sampling a string, DEC GKS takes the first characters in the entered text string, including any initial prompt, up to the number of characters specified by the size of the application's buffer. If the size of the entered string is larger than the number of characters placed in the application's buffer, DEC GKS performs the following tasks:

- Removes the sampled string (the size of the application's buffer) from the device's buffer.
- Places the the sampled string in the application's buffer.
- Leaves any remaining characters in the device's buffer. You need to call GKS$SAMPLE_STRING again to access the remaining characters.

Syntax

GKS$SAMPLE_STRING *(workstation_id, device_number, string_buffer, string_size, total_string_size)*

GSMST *(workstation_id, dev_num, num_char, cstring)*

GSMST - Subset *(workstation_id, dev_num, num_char, cstring)*

gsamplestring *(workstation_id, dev, response)*

Arguments

workstation_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

string_buffer

data type: **string**
access: **write-only**
mechanism: **by descriptor**

This is the argument to which DEC GKS writes the input character string. This is the *application's* buffer.

string_size

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the number of bytes in the string accepted by the input sample.

total_string_size

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the total number of characters in the device's buffer, in bytes. If this argument's value is greater than the value

of string_size, you may wish to call GKS$SAMPLE_STRING again to obtain the characters remaining in the device's buffer.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
142	GKS$_ERROR_142	Input device is not in SAMPLE mode in routine ****

Program Example

Example 8–13 illustrates the use of the function GKS$SAMPLE_STRING. Following the program example, Figures 8–24 through 8–27 illustrate the program's effect on a VT241 workstation.

Example 8–13: Using a String Logical Input Device in Sample Mode

```
C     This program initializes and samples string input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, DATA_RECORD( 2 ),
     * PROMPT_ECHO_TYPE, ERROR_STATUS, BUFFER_LENGTH,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE, INPUT_STATUS, DEVICE_NUM, STRING_SIZE,
     * CUR_POSITION, TOTAL_STRING_SIZE, INCR, INCR2,
     * EVENT_FLAG
      REAL*8 TIME
      REAL ECHO_AREA( 4 ), START_X, START_Y, X_VECTOR, Y_VECTOR,
     * LARGER
      CHARACTER*31 INITIAL_STRING, STRING_BUFFER
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, LARGER / 0.03 /

C     First element in the data record is length of the buffer that
C     contains the input string.
      EQUIVALENCE( DATA_RECORD( 1 ), BUFFER_LENGTH )
      EQUIVALENCE( DATA_RECORD( 2 ), CUR_POSITION )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      RECORD_BUFFER_LENGTH = 8
      CALL GKS$INQ_STRING_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, INITIAL_STRING,
     * STRING_SIZE, PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

C     Change the current input values.
(1)   ECHO_AREA( 1 ) = 437
      BUFFER_LENGTH = 31
      CUR_POSITION = 1

C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_STRING( WS_ID, DEVICE_NUM, ' ',
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )
```

(continued on next page)

Example 8–13 (Cont.): Using a String Logical Input Device in Sample Mode

```
C     Tell the user what is happening.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.05, 0.95, 'Every 20 seconds, type a string.')
      CALL GKS$TEXT( 0.05, 0.90, 'I will use them in my design.')

❷     CALL GKS$SET_STRING_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_SAMPLE, GKS$K_ECHO )

C     Set the initial text settings and attributes.
      START_X = 0.05
      START_Y = 0.70
      X_VECTOR = 0.0
      Y_VECTOR = 1.0

C     Obtain an event flag for the timer.
      CALL LIB$GET_EF( EVENT_FLAG )

C     Do for three lines.
      DO 200 INCR = 1, 3, 1

      IF ( INCR .NE. 1 ) THEN

C     Give the user a 20-second break.
❸        CALL SYS$BINTIM( '0 :00:20', TIME )
         CALL SYS$SETIMR( %VAL( EVENT_FLAG ), -( TIME ),,)
         CALL SYS$WAITFR( %VAL( EVENT_FLAG ) )

C        Sample the string.
❹        CALL GKS$SAMPLE_STRING( WS_ID, DEVICE_NUM,
      *  STRING_BUFFER, STRING_SIZE, TOTAL_STRING_SIZE )
      ELSE
C        Provide the first string.
         STRING_BUFFER = 'I''ll give you the first string.'
      ENDIF

C     Create a design with a text string.
❺     DO 300 INCR2 = 1, 3, 1
         CALL GKS$SET_TEXT_UPVEC( X_VECTOR, Y_VECTOR )
         CALL GKS$TEXT( START_X, START_Y,
      *       STRING_BUFFER )
         X_VECTOR = X_VECTOR + 0.1
         Y_VECTOR = Y_VECTOR - 0.1
300   CONTINUE
```

(continued on next page)

Example 8–13 (Cont.): Using a String Logical Input Device in Sample Mode

```
C     Reset variables.
      START_Y = START_Y - 0.01
      STRING_BUFFER = ' '

200   CONTINUE

C     Free the event flag used for the timer.
      CALL LIB$FREE_EF( EVENT_FLAG )
C     Turn off the sample prompt.
❻     CALL GKS$SET_STRING_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ Changing the value of variable ECHO_AREA(1) widens the echo area so that the user can enter a larger string than the one which the default area allows.

❷ The call to GKS$SET_STRING_MODE sets the input operating mode to sample. At this point in the program, the string prompt appears on the workstation surface and the user can change the measure of the device.

❸ This code creates a 20-second timer that allows the user to enter and alter a character string. For more information concerning these function calls, refer to the *Introduction to VMS System Routines* and to the *VAX/VMS Run-Time Library Routines Reference Manual.*

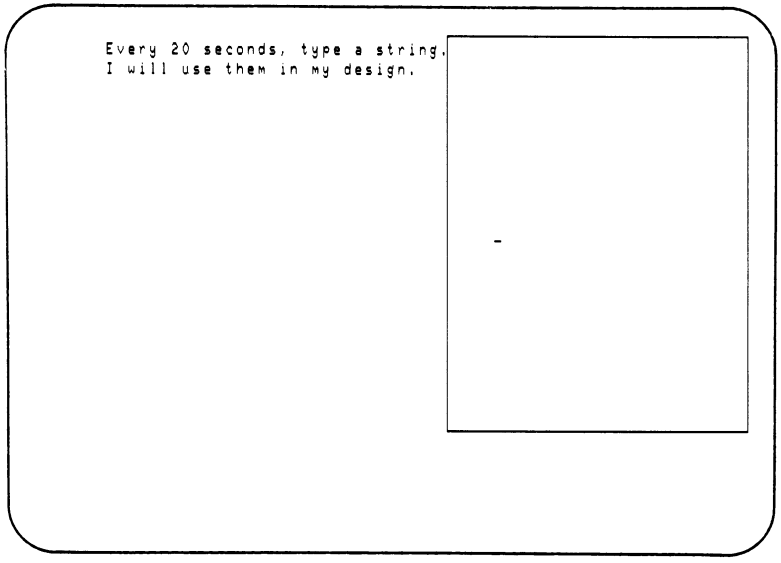
❹ This call to GKS$SAMPLE_STRING retrieves the current input value (without the user having to trigger the device). This loop requires two strings from the user. The program provides the first string.

❺ This code outputs the strings by adjusting the character-up vector. This creates a pinwheel design on the workstation surface.

❻ The call to GKS$SET_STRING_MODE returns the logical input device to request mode. At this point, the device handler removes the string prompt from the workstation surface and the user can no longer enter input.

Figure 8–24 illustrates the surface of the VT241 just after the input mode is set. Figure 8–25 illustrates the surface of the VT241 after the user types a string (note that if the user presses RETURN, nothing happens—the application decides when to sample input). Figure 8–26 illustrates the surface of the VT241 after the program accepts the first string. Notice that the user need only enter and alter the string (without triggering) and the program samples the current string accordingly. Figure 8–27 illustrates the surface of the VT241 after the user entered all strings.

Figure 8–24: The String Logical Input Device in Sample Mode—VT241



ZK-5823-HC

Figure 8–25: The String Logical Input Device in Sample Mode—VT241

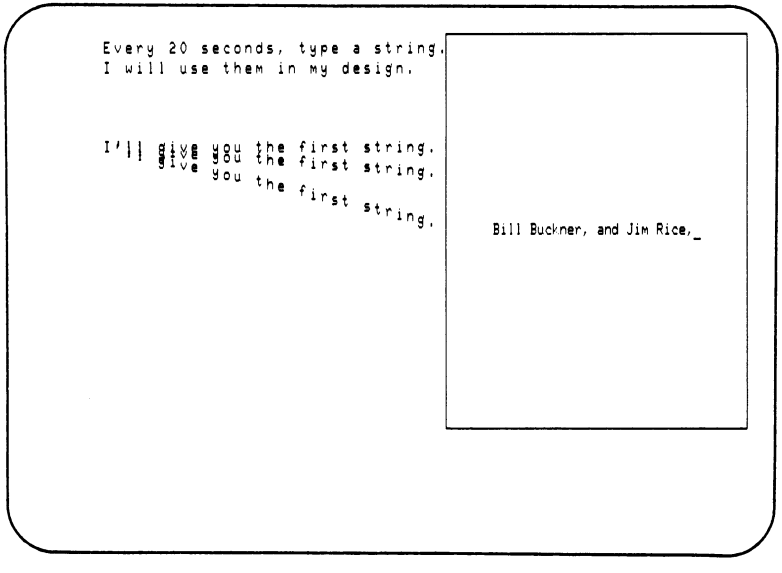
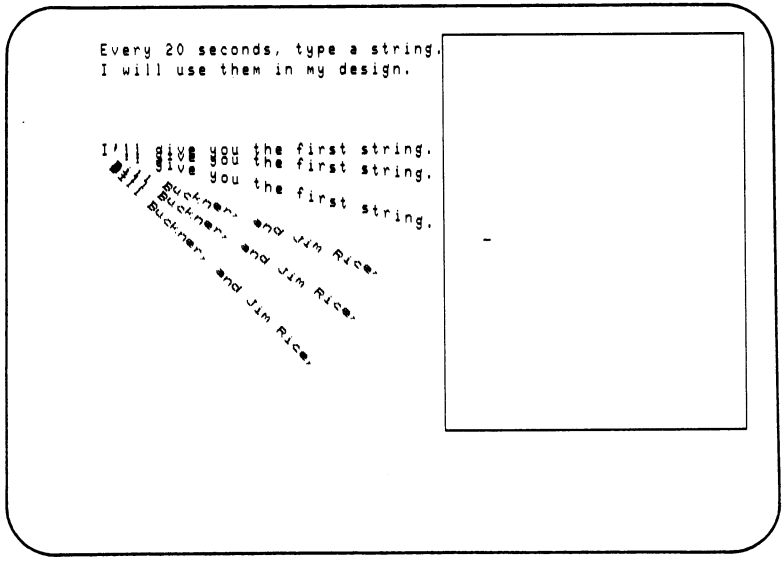
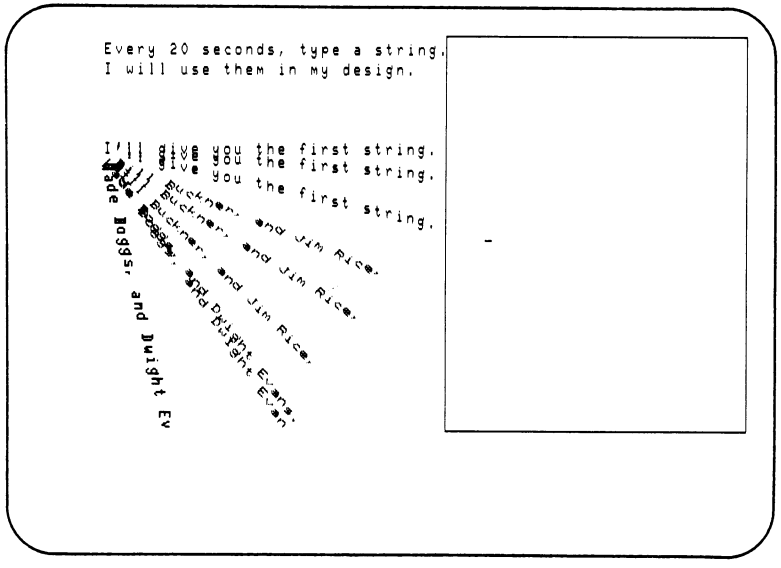


Figure 8–26: The String Logical Input Device in Sample Mode—VT241



ZK-5813-HC

Figure 8–27: The String Logical Input Device in Sample Mode—VT241



ZK-5814-HC

SAMPLE STROKE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SAMPLE_STROKE writes the current measure of the specified stroke logical input device to the corresponding output arguments.

When activating stroke input, the following two buffers exist:

- The application's stroke buffer, whose size you specify when you pass the buffer argument by descriptor to GKS$SAMPLE_STROKE.
- The logical input device's stroke buffer, whose size you can specify in the call to GKS$INIT_STROKE.

When sampling stroke input, DEC GKS accepts any initial stroke points and translates entered points according to the current normalization transformation. DEC GKS can accept points up to the number specified by the size of the application's buffer. If the size of the entered stroke is larger than the number of stroke points placed in the application's buffer, DEC GKS performs the following tasks:

- Removes the sampled stroke (the size of the application's buffer) from the device's buffer.
- Places the the sampled stroke in the application's buffer.
- Leaves any remaining points in the device's buffer. You need to call GKS$SAMPLE_STROKE again to access the remaining characters.

Syntax

GKS$SAMPLE_STROKE *(workstation_id, device_number, transformation_number, num_entered_points, stroke_buffer_x, stroke_buffer_y, stroke_size_x, stroke_size_y)*

GSMSK *(workstation_id, dev_num, max_pts, xform, num_pts, px, py)*

gsamplestroke *(workstation_id, dev, response)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

transformation_number

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This is the argument to which DEC GKS writes the number of the normalization transformation used to translate the input points to world coordinate points.

num_entered_points

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This is the argument to which DEC GKS writes the total number of points in the stroke entered by the user.

stroke_buffer_x
stroke_buffer_y

data type:	**array (real)**
access:	**write-only**
mechanism:	**by descriptor**

These are the arguments to which DEC GKS writes the X and Y world coordinate values of the accepted stroke. These arguments are the application's stroke buffer.

stroke_size_x
stroke_size_y

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

These are the arguments to which DEC GKS writes the number of stroke points actually accepted in the application buffer.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
142	GKS$_ERROR_142	Input device is not in SAMPLE mode in routine ****

Program Example

Example 8–14 illustrates the use of the function GKS$SAMPLE_STROKE. Following the program example, Figures 8–28 through 8–33 illustrate the program's effect on a VT241 workstation.

Example 8–14: Using a Stroke Logical Input Device in Sample Mode

```
C     This program initializes and samples stroke input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, DATA_RECORD( 6 ), BUFFER_SIZE,
     * DIMENSION, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * TRANSFRM, NUM_ENTERED_POINTS, INPUT_MODE, ECHO_FLAG,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE, INPUT_STATUS, DEVICE_NUM,
     * I, EDIT_POSITION, ATTS_FLAG, TEXT, INCR, INCR2, EVENT_FLAG,
     * RET_SIZE_BUF( 3 ), RET_SIZE_X, RET_SIZE_Y
      REAL ECHO_AREA( 4 ), STROKE_X( 50 ),
     * STROKE_Y( 50 ), X_INT, Y_INT, TIME_INT,
     * STROKE_BUFFER_X( 3, 50 ), STROKE_BUFFER_Y( 3, 50 ),
     * LARGER
      REAL*8 TIME
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, TEXT / 1 /,
     * LARGER / 0.03 /

C     First element in the data record is the buffer size.
      EQUIVALENCE( DATA_RECORD( 1 ), BUFFER_SIZE)
      EQUIVALENCE( DATA_RECORD( 2 ), EDIT_POSITION)
      EQUIVALENCE( DATA_RECORD( 3 ), X_INT)
      EQUIVALENCE( DATA_RECORD( 4 ), Y_INT)
      EQUIVALENCE( DATA_RECORD( 5 ), TIME_INT)
      EQUIVALENCE( DATA_RECORD( 6 ), ATTS_FLAG)

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      RECORD_BUFFER_LENGTH = 24
      CALL GKS$INQ_STROKE_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, DIMENSION, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, TRANSFRM, NUM_ENTERED_POINTS,
     * STROKE_X, STROKE_Y, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH, RECORD_SIZE )

C     Allow a buffer that is large enough.
C      TRANSFRM = 0

C     By specifying to DEC GKS to use the current attributes flag, you
C     need to pass the 24 byte data record instead of the 52 byte record.
      ATTS_FLAG = GKS$K_ACF_CURRENT
```

(continued on next page)

Example 8–14 (Cont.): Using a Stroke Logical Input Device in Sample Mode

```
C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_STROKE( WS_ID, DEVICE_NUM,
     * NUM_ENTERED_POINTS, STROKE_X, STROKE_Y, TRANSFRM,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )

❶     CALL GKS$SET_STROKE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_SAMPLE, GKS$K_ECHO )

C     Tell the user how many sets of stroke points to enter.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.05, 0.95,
     * 'Every 20 seconds, enter points.')
      CALL GKS$TEXT( 0.05, 0.90,
     * 'I''ll show the three fill areas.')

C     Obtain an event flag for the timer.
      CALL LIB$GET_EF( EVENT_FLAG )
C     Do for three sets of stroke points.
      DO 200 INCR = 1, 3, 1

C     Give the user a 20-second break.
❷     CALL SYS$BINTIM( '0 :00:20', TIME )
      CALL SYS$SETIMR( %VAL( EVENT_FLAG ), -( TIME ),,)
      CALL SYS$WAITFR( %VAL( EVENT_FLAG ) )

C     Sample the stroke.
❸     CALL GKS$SAMPLE_STROKE( WS_ID, DEVICE_NUM,
     * TRANSFRM, NUM_ENTERED_POINTS, %DESCR( STROKE_X ),
     * %DESCR( STROKE_Y ), RET_SIZE_X, RET_SIZE_Y )

❹     RET_SIZE_BUF( INCR ) = MIN( RET_SIZE_X, RET_SIZE_Y )

C     Put the strokes in a buffer.
      DO 300 INCR2 = 1, RET_SIZE_BUF( INCR ), 1
      STROKE_BUFFER_X( INCR, INCR2 ) = STROKE_X( INCR2 )
      STROKE_BUFFER_Y( INCR, INCR2 ) = STROKE_Y( INCR2 )
300   CONTINUE

200   CONTINUE
```

(continued on next page)

Example 8–14 (Cont.): Using a Stroke Logical Input Device in Sample Mode

```
C     Free the event flag used for the timer.
      CALL LIB$FREE_EF( EVENT_FLAG )

C     Get rid of the stroke prompt...
❺     CALL GKS$SET_STROKE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

C     Present the corresponding fill areas.  Press RETURN when you are
C     ready to view the next fill area.
❻     CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_ALWAYS )
      CALL GKS$TEXT( 0.05, 0.95,
     * 'Here are the fill areas.')
      CALL GKS$CREATE_SEG( TEXT )
      CALL GKS$TEXT( 0.05, 0.90,
     * 'Press RETURN when ready.')
      CALL GKS$CLOSE_SEG()

      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )

      DO 400 INCR = 1, 3, 1

C     Put the current stroke in the temporary buffer.
      DO 500 INCR2 = 1, RET_SIZE_BUF( INCR ), 1
      STROKE_X( INCR2 ) = STROKE_BUFFER_X( INCR, INCR2 )
      STROKE_Y( INCR2 ) = STROKE_BUFFER_Y( INCR, INCR2 )
500   CONTINUE

      CALL GKS$FILL_AREA( RET_SIZE_BUF( INCR ),
     * STROKE_X, STROKE_Y )
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
      IF ( INCR .EQ. 3 ) THEN
         CALL GKS$DELETE_SEG( TEXT )
      ENDIF
      CALL GKS$REDRAW_SEG_ON_WS( WS_ID )

400   CONTINUE

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The call to GKS$SET_STROKE_MODE sets the input operating mode to sample. At this point in the program, the stroke prompt appears on the workstation surface and the user can enter or alter stroke points.

❷ This code creates a 20-second timer that allows the user to enter and alter a character string. For more information concerning these function calls, refer to the *Introduction to VMS System Routines*, and to the *VAX/VMS Run-Time Library Routines Reference Manual*.

❸ This call to GKS$SAMPLE_STROKE retrieves the current input value (without the user having to trigger the device). The loop allows the user to enter three sets of stroke points.

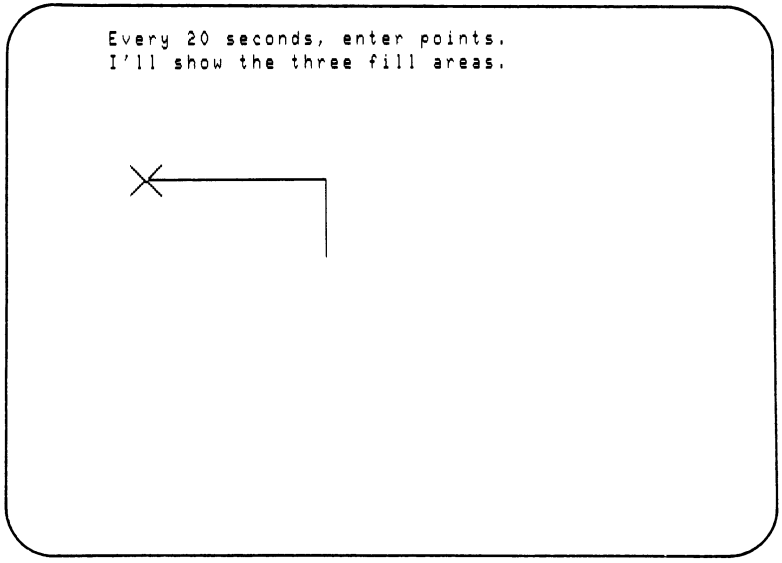
❹ This code sets the correct size of the buffer and stores the stroke points in a two-dimensional array.

❺ The call to GKS$SET_STROKE_MODE returns the logical input device to request mode. At this point, the device handler removes the stroke prompt from the workstation surface and the user can no longer enter input.

❻ This code uses each of the sets of stroke points to create and display a fill area.

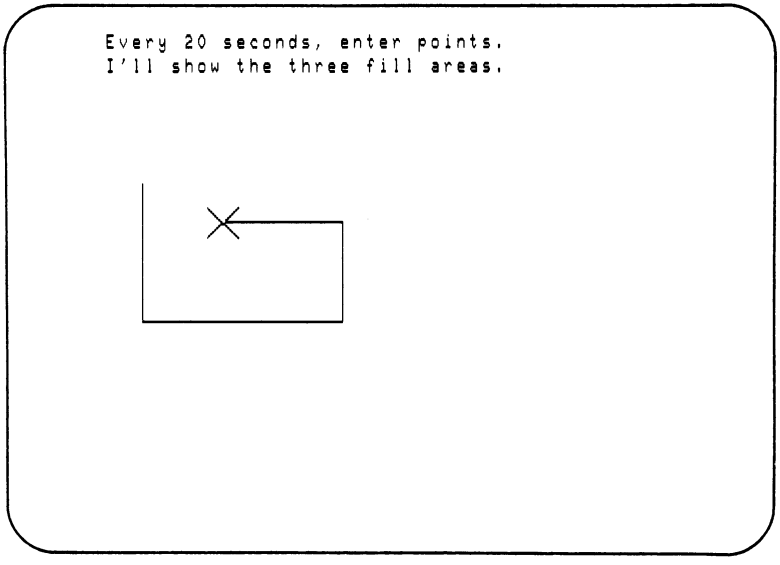
Figure 8–28 illustrates the surface of the VT241 just before the program accepts the first set of stroke points. Notice that the user need only enter points (without triggering) and the program accepts the current set. Figures 8–29 and 8–30 illustrate the remaining sets of stroke points entered by the user. Figures 8–31 through 8–33 show the fill areas generated from the set of entered stroke points.

Figure 8–28: The Stroke Logical Input Device in Sample Mode—VT241



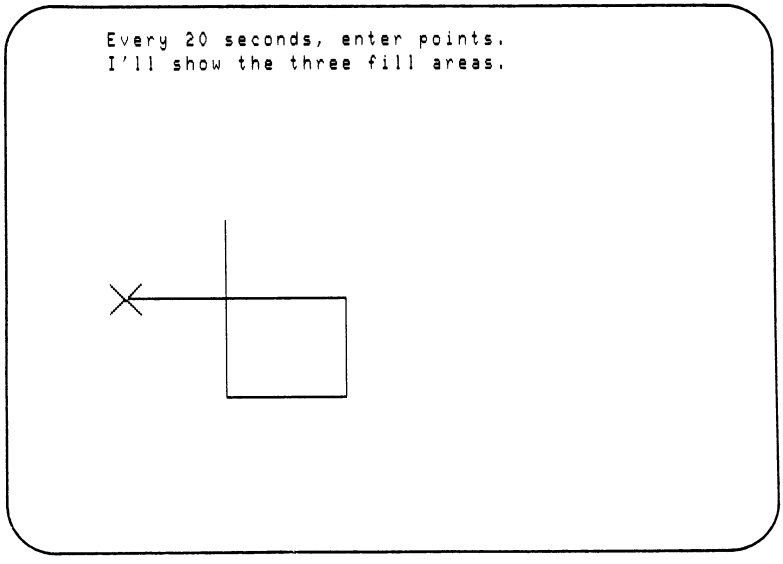
ZK-5827-HC

Figure 8–29: The Stroke Logical Input Device in Sample Mode—VT241



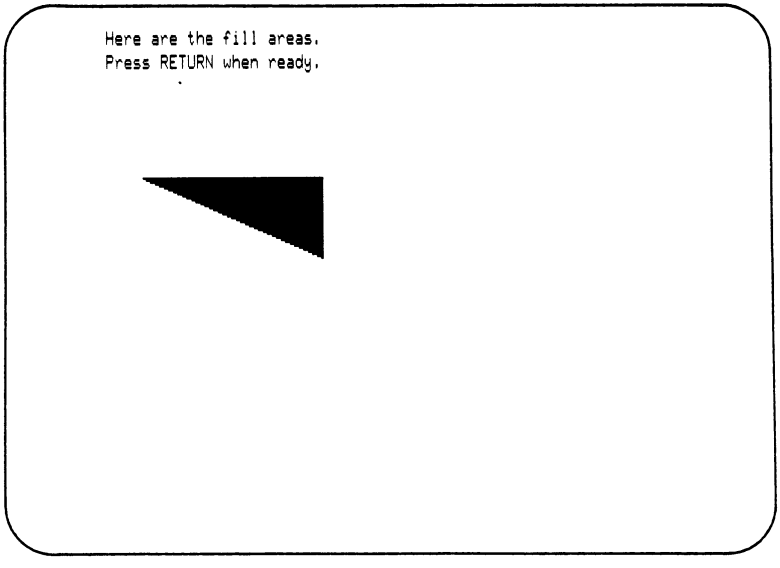
ZK-5828-HC

Figure 8–30: The Stroke Logical Input Device in Sample Mode—VT241



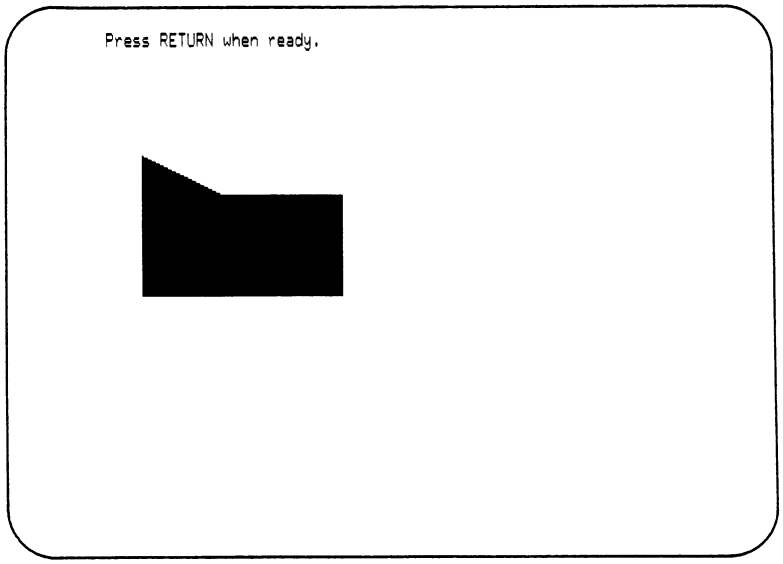
ZK-5829-HC

Figure 8–31: The Stroke Logical Input Device in Sample Mode—VT241



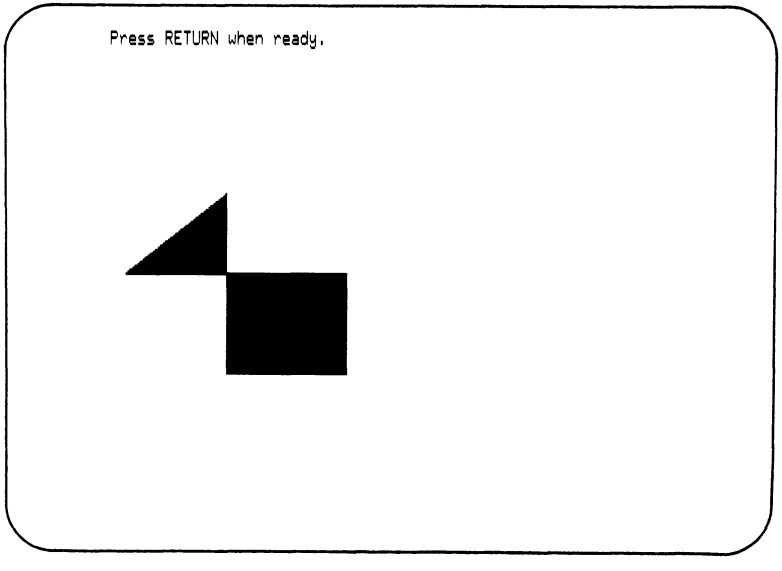
ZK-5830-HC

Figure 8–32: The Stroke Logical Input Device in Sample Mode—VT241



ZK-5831-HC

Figure 8–33: The Stroke Logical Input Device in Sample Mode—VT241



ZK-5832-HC

SAMPLE VALUATOR

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SAMPLE_VALUATOR writes the current measure of the specified valuator logical input device to the corresponding output argument.

Syntax

GKS$SAMPLE_VALUATOR *(workstation_id, device_number, real_value)*

GSMVL *(workstation_id, dev_num, value)*

gsampleval *(workstation_id, dev, response)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier specified in a previous call to GKS$OPEN_WS.

device_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the device number that differentiates logical devices of the same class operating on the same workstation.

real_value

data type: **real**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the current measure of the valuator device.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
142	GKS$_ERROR_142	Input device is not in SAMPLE mode in routine ****

Program Example

Example 8–15 illustrates the use of the function GKS$SAMPLE_VALUATOR. Following the program example, Figures 8–34 through 8–36 illustrate the program's effect on a VT241 workstation.

Example 8–15: Using a Valuator Logical Input Device in Sample Mode

```
C     This program initializes and samples valuator input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH, RECORD_SIZE,
     * INPUT_STATUS, DEVICE_NUM, BOX, INCR, DUMMY_INTEGER,
     * NEW_FRAME_FLAG
      REAL ECHO_AREA( 4 ), DATA_RECORD( 2 ), UPPER_LIMIT,
     * LOWER_LIMIT, VALUE, BOX_X( 5 ), BOX_Y( 5 ), LARGER,
     * XFORM_MATRIX( 6 )
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, BOX / 1 /,
     * LARGER / 0.03 /
      DATA BOX_X / 0.4, 0.6, 0.6, 0.4, 0.4 /
      DATA BOX_Y / 0.4, 0.4, 0.6, 0.6, 0.4 /

C     The elements in the data record are the upper and lower limits.
      EQUIVALENCE( DATA_RECORD( 1 ), LOWER_LIMIT )
      EQUIVALENCE( DATA_RECORD( 2 ), UPPER_LIMIT )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      RECORD_BUFFER_LENGTH = 8
      CALL GKS$INQ_VALUATOR_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, VALUE,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

      VALUE = 1.0
      UPPER_LIMIT = 2.0
      LOWER_LIMIT = 0.001

C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_VALUATOR( WS_ID, DEVICE_NUM,
     * VALUE, PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )

(1)   CALL GKS$SET_VALUATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_SAMPLE, GKS$K_ECHO )
```

(continued on next page)

Example 8–15 (Cont.): Using a Valuator Logical Input Device in Sample Mode

```
❷       CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )
        CALL GKS$SET_TEXT_HEIGHT( LARGER )

        CALL GKS$CREATE_SEG( BOX )
        CALL GKS$FILL_AREA( 5, BOX_X, BOX_Y )
        CALL GKS$CLOSE_SEG()

C       Inform the user about the task.
        CALL GKS$TEXT( 0.05, 0.95,
     * 'Alter the box''s size.' )
        CALL GKS$TEXT( 0.05, 0.90,
     * 'To stop, set the value to 2.0.' )

        DO WHILE ( VALUE .NE. 2.0 )

❸       CALL GKS$SAMPLE_VALUATOR( WS_ID, DEVICE_NUM,
     * VALUE )

C       Scale the segment according to the VALUE argument.
        CALL GKS$EVAL_XFORM_MATRIX( 0.5, 0.5, 0.0, 0.0, 0.0,
     * VALUE, VALUE, GKS$K_COORDINATES_WC, XFORM_MATRIX )

        IF ( VALUE .NE. 1.0 ) THEN
           CALL GKS$SET_SEG_XFORM( BOX, XFORM_MATRIX )
           CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )
        ENDIF
        ENDDO

C       Turn off the sample prompt.
❹       CALL GKS$SET_VALUATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

        CALL GKS$DEACTIVATE_WS( WS_ID )
        CALL GKS$CLOSE_WS( WS_ID )
        CALL GKS$CLOSE_GKS()
        END
```

The following numbers correspond to the numbers in the previous example:

❶ The call to GKS$SET_VALUATOR_MODE sets the input operating mode to sample. At this point in the program, the valuator prompt appears on the workstation surface and the user can change the measure of the device.

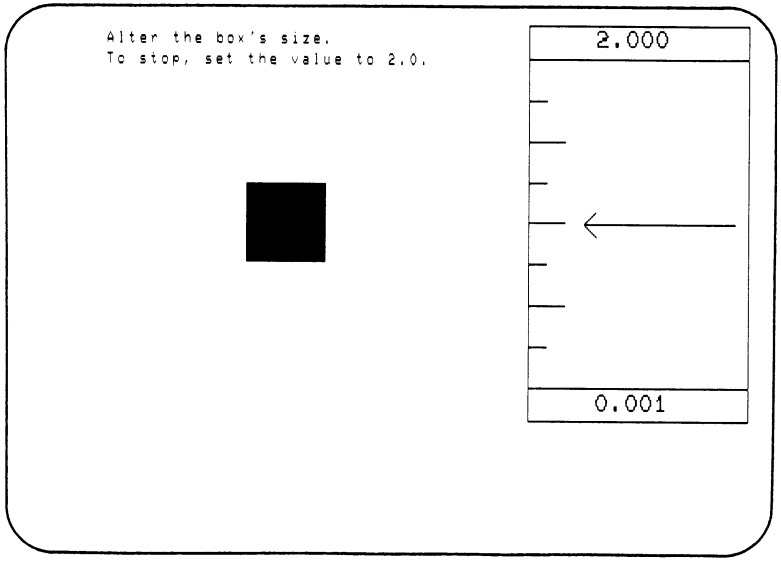
❷ This code creates a segment containing a square fill area. The program scales this box according to the sampled value of the valuator device.

❸ This call to GKS$SAMPLE_VALUATOR retrieves the current input value (without the user having to trigger the device). The loop ends when the user moves the prompt to the value 2.0.

❹ The call to GKS$SET_VALUATOR_MODE returns the logical input device to request mode. At this point, the device handler removes the valuator prompt from the workstation surface and the user can no longer enter input.

Figure 8–34 illustrates the surface of the VT241 when the input mode is set. Figure 8–35 illustrates the surface of the VT241 when the user moves the prompt. Notice that the user need only move the prompt to another value (without triggering) and the program scales the segment accordingly. Figure 8–36 illustrates the surface of the VT241 when the program ends.

Figure 8–34: The Valuator Logical Input Device in Sample Mode—VT241



ZK-5840-HC

Figure 8–35: The Valuator Logical Input Device in Sample Mode—VT241

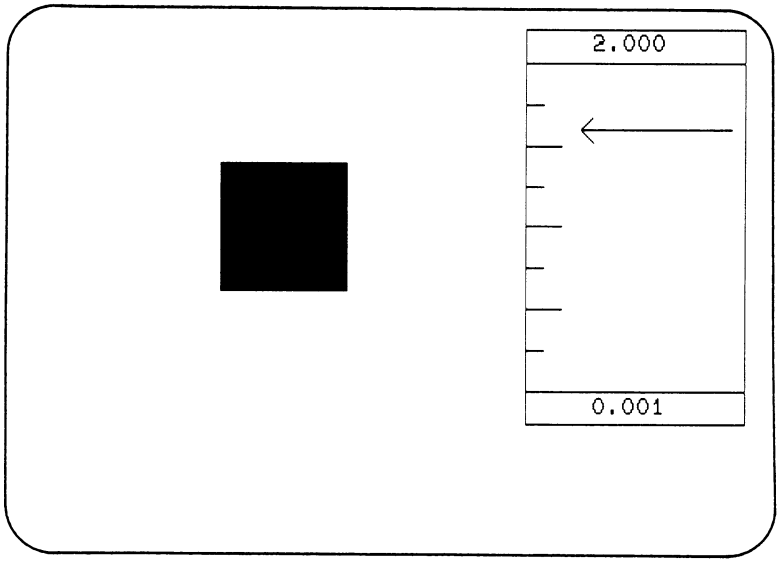
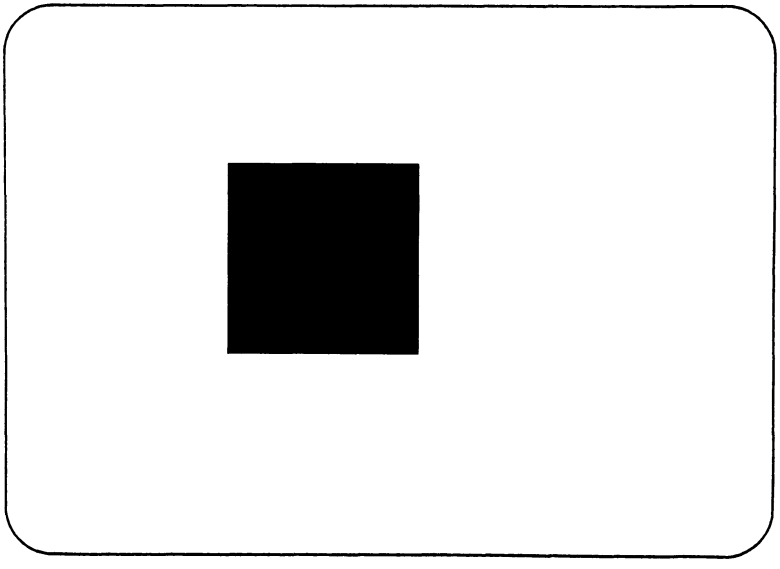


Figure 8–36: The Valuator Logical Input Device in Sample Mode—VT241



ZK-5841-HC

Obtaining Input in Event Mode

This section describes the functions used to remove, read, and flush input reports from the event queue. You should use GKS$AWAIT_EVENT to fetch a report from the queue and to place it in the *current event report* entry in the DEC GKS state list. Also, GKS$AWAIT_EVENT writes the logical input class of the device that accepted the current report to one of its output arguments. To read the input information from the current event report, call the appropriate GKS$GET_class function. If you call one of the GKS$GET_class functions for an event in the current event report that was not generated by a device of the corresponding class, you generate an error.

Remember that repeated calls to one of the functions GKS$GET_LOCATOR, GKS$GET_STROKE, and so forth, will write the same values to the output arguments since these functions always obtain information from the current event report. The current event report does not change unless you call GKS$AWAIT_EVENT to fetch another report from the queue. Once you do this, a subsequent call to one of the GKS$GET_class functions obtains new input values.

See Section 8.5 for information concerning event operating modes.

This section describes the following functions:

- GKS$AWAIT_EVENT
- GKS$FLUSH_DEVICE_EVENTS
- GKS$GET_CHOICE
- GKS$GET_LOCATOR
- GKS$GET_PICK
- GKS$GET_STRING
- GKS$GET_STROKE
- GKS$GET_VALUATOR

AWAIT EVENT

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$AWAIT_EVENT examines the input queue for all input devices.

DEC GKS searches the input queue for an event and, if the input queue is empty, suspends the application program until either of the following happens:

- An event appears on the input queue.
- The time period specified in the time-out argument expires.

If you specify zero (0.0) as the time-out argument, DEC GKS checks the input queue immediately without suspending the application.

When GKS$AWAIT_EVENT checks the event input queue, its subsequent action depends on the state of the queue. If the queue contains reports, this function performs the following tasks:

- Removes the oldest event report from the queue.
- Writes information to the current event report entry in the DEC GKS state list.
- Writes the event's workstation identifier, input class, and logical device number to its corresponding output arguments.

If the time-out period has expired, and if GKS$AWAIT_EVENT finds the queue to be empty, this function writes GKS$K_INPUT_CLASS_NONE to its input class argument.

If you generate the queue-overflow error, this function still performs its task as described. See Section 8.5.3.3 for information concerning input-queue overflow.

Syntax

GKS$AWAIT_EVENT *(time_out, workstation_id, input_class, device_number)*

GWAIT *(time_out, ws_id, in_class, dev_num)*

gawaitevent *(timeout, event)*

Arguments

time_out

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

This argument is the amount of time to wait for an event to appear in the input queue. This argument is specified in the following format:

ss.hh

Where *ss* is seconds and *hh* is hundreds of a second. This argument cannot be negative and cannot be larger than 356,400 seconds (99 hours).

If this argument is zero (0.0), this function allows application execution to continue and either removes the oldest event or, if there are no events in the queue, returns GKS$K_INPUT_CLASS_NONE to the input_class argument.

workstation_id

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This argument is the workstation identifier that corresponds to the logical input device that accepted the event.

input_class

data type: **integer**
access: **write-only**
mechanism: **by reference**

This argument is the input class that corresponds to the logical input device that accepted the event. This argument can be any of the following:

Value	Constant	Description
0	GKS$K_INPUT_CLASS_NONE	Input queue is empty.
1	GKS$K_INPUT_CLASS_LOCATOR	Event from a locator device.
2	GKS$K_INPUT_CLASS_STROKE	Event from a stroke device.
3	GKS$K_INPUT_CLASS_VALUATOR	Event from a valuator device.
4	GKS$K_INPUT_CLASS_CHOICE	Event from a choice device.
5	GKS$K_INPUT_CLASS_PICK	Event from a pick device.
6	GKS$K_INPUT_CLASS_STRING	Event from a string device.

device_number

data type: **integer**
access: **write-only**
mechanism: **by reference**

This argument is the device number that corresponds to the logical input device that accepted the event.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
147	GKS$_ERROR_147	Input queue has overflowed in routine ****
151	GKS$_ERROR_151	Timeout is invalid in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–3.

FLUSH DEVICE EVENTS

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$FLUSH_DEVICE_EVENTS removes all events generated by one class of input device from the input queue. This function performs its task even if it generates the queue-overflow error message.

See Section 8.5.3 for information concerning the flushing of the device queue, and see Section 8.5.3.3 for information concerning input-queue overflow.

Syntax

GKS$FLUSH_DEVICE_EVENTS *(workstation_id, input_class, device_number)*

GFLUSH *(workstation_id, in_class, dev_num)*

gflushevents *(workstation_id,class,dev)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the workstation identifier that corresponds to the logical input device of the events to be removed from the input queue.

input_class

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the input class that corresponds to the logical input device of

the events to be removed from the input queue. This argument can be any of the following:

Value	Constant	Description
1	GKS$K_INPUT_CLASS_LOCATOR	Event from a locator device.
2	GKS$K_INPUT_CLASS_STROKE	Event from a stroke device.
3	GKS$K_INPUT_CLASS_VALUATOR	Event from a valuator device.
4	GKS$K_INPUT_CLASS_CHOICE	Event from a choice device.
5	GKS$K_INPUT_CLASS_PICK	Event from a pick device.
6	GKS$K_INPUT_CLASS_STRING	Event from a string device.

device_number

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the device number that corresponds to the logical input device of the events to be removed from the queue.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****

Error Number	Completion Status Code	Message
38	GKS$_ERROR_38	Specified workstation is neither of category INPUT nor of category OUTIN in routine ****
140	GKS$_ERROR_140	Specified input device is not present on workstation in routine ****
147	GKS$_ERROR_147	Input queue has overflowed in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–5.

GET CHOICE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$GET_CHOICE obtains information from the current event report entry in the DEC GKS state list and writes the choice status and choice value to the output arguments.

Syntax

GKS$GET_CHOICE *(input_status, choice_value)*

GGTCH *(in_status, ch_num)*

ggetchoice *(response)*

Arguments

input_status

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This is the argument to which DEC GKS writes the status of the input process for the current event report. This argument can be either of the following values or constants:

Value	Constant	Description
1	GKS$K_STATUS_OK	Input obtained.
2	GKS$K_STATUS_NOCHOICE	Triggered without choosing.

choice_value

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer representing the user's choice for the current event report.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
150	GKS$_ERROR_150	No input value of the correct class is in the current event report in routine ****

Program Example

Example 8–16 illustrates the use of the function GKS$GET_CHOICE.

Example 8–16: Using a Choice Logical Input Device in Event Mode

```
C     This program initializes and accepts choice events from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, DATA_RECORD( 3 ), NUM_CHOICES, SIZES( 3 ),
     * ADDRESSES( 3 ), PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH, RECORD_SIZE,
     * INPUT_STATUS, INITIAL_CHOICE, DEVICE_NUM, INPUT_CHOICE,
     * INITIAL_STATUS, NUM_POINTS, COLOR1, COLOR2, COLOR3,
     * CLASS
      REAL ECHO_AREA( 4 ), PX( 4 ), PY( 4 ), LARGER
      DATA PX / 0.05, 0.1, 0.075, 0.05 /
      DATA PY / 0.85, 0.85, 0.80, 0.85 /
      DATA NUM_POINTS / 4 /, COLOR1 / 1 /, COLOR2 / 2 /,
     * COLOR3 / 3 /, LARGER / 0.03 /

      CHARACTER*80 CURRENT_STRINGS( 3 )

      DATA WS_ID / 1 /, DEVICE_NUM / 1 /

C     First element in the data record is the number of choices.
      EQUIVALENCE( DATA_RECORD(1), NUM_CHOICES )
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Establish the size of the record buffer: 12 bytes.
      RECORD_BUFFER_LENGTH = 12

C     The second element in the VT241 choice data record is the pointer to
C     the array containing sizes of each choice character string. You need
C     to initialize the pointer so that the array can be initialized.
      DATA_RECORD( 2 ) = %LOC( SIZES(1) )

C     The third element in the VT241 choice data record is the pointer to the
C     array containing the pointers to the strings to be used. You need
C     to initialize the pointer so that the array can be initialized.
      DATA_RECORD( 3 ) = %LOC( ADDRESSES(1) )
      ADDRESSES( 1 ) = %LOC( CURRENT_STRINGS( 1 ) )
      ADDRESSES( 2 ) = %LOC( CURRENT_STRINGS( 2 ) )
      ADDRESSES( 3 ) = %LOC( CURRENT_STRINGS( 3 ) )
```

(continued on next page)

Example 8–16 (Cont.): Using a Choice Logical Input Device in Event Mode

```
C     Inquire about the default values.
      NUM_CHOICES = 3
      CALL GKS$INQ_CHOICE_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, INITIAL_STATUS,
     * INITIAL_CHOICE, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH, RECORD_SIZE )

C     Set the initial status.
      INITIAL_STATUS = GKS$K_STATUS_OK

C     Establish sizes of prompt strings...
      SIZES( 1 ) = 4
      SIZES( 2 ) = 5
      SIZES( 3 ) = 4

C     Establish locations of prompt strings...
      ADDRESSES( 1 ) = %LOC( 'Pink' )
      ADDRESSES( 2 ) = %LOC( 'Green' )
      ADDRESSES( 3 ) = %LOC( 'Blue' )

C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_CHOICE( WS_ID, DEVICE_NUM,
     * INITIAL_STATUS, INITIAL_CHOICE, PROMPT_ECHO_TYPE,
     * ECHO_AREA, DATA_RECORD, RECORD_BUFFER_LENGTH )

❶     CALL GKS$SET_CHOICE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_EVENT, GKS$K_ECHO )

C     Initialize color indexes.
      CALL GKS$SET_COLOR_REP( WS_ID, COLOR1, 0.6258, 0.2142,
     * 0.2142 )
      CALL GKS$SET_COLOR_REP( WS_ID, COLOR2, 0.1400, 1.000,
     * 0.1400 )
      CALL GKS$SET_COLOR_REP( WS_ID, COLOR3, 0.0, 0.0,
     * 0.8400 )
      CALL GKS$SET_FILL_COLOR_INDEX( COLOR1 )
      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )
```

(continued on next page)

Example 8–16 (Cont.): Using a Choice Logical Input Device in Event Mode

```
C     Tell the user how to change colors.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.05, 0.95, 'Move the arrow keys to')
      CALL GKS$TEXT( 0.05, 0.90, 'change the triangle colors.')

C     Do until the surface is full.
      DO WHILE ( PX( 2 ) .LT. 0.95 )

C     Check the event queue.
      CALL GKS$AWAIT_EVENT( 0.0, WS_ID, CLASS, DEVICE_NUM )

(2)   IF ( CLASS .NE. GKS$K_INPUT_CLASS_NONE ) THEN
         CALL GKS$GET_CHOICE( INPUT_STATUS, INPUT_CHOICE )
      ENDIF

C     Depending on the event values, change the color.
      IF ( INPUT_CHOICE .EQ. 1 ) THEN
         CALL GKS$SET_FILL_COLOR_INDEX( COLOR1 )
      ENDIF
      IF ( INPUT_CHOICE .EQ. 2 ) THEN
         CALL GKS$SET_FILL_COLOR_INDEX( COLOR2 )
      ENDIF
      IF ( INPUT_CHOICE .EQ. 3 ) THEN
         CALL GKS$SET_FILL_COLOR_INDEX( COLOR3 )
      ENDIF

C     Draw the triangles.
(3)   CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )

C     Adjust the position of the triangles.
      PY( 1 ) = PY( 1 ) - 0.06
      PY( 2 ) = PY( 2 ) - 0.06
      PY( 3 ) = PY( 3 ) - 0.06
      PY( 4 ) = PY( 4 ) - 0.06
```

(continued on next page)

Example 8–16 (Cont.): Using a Choice Logical Input Device in Event Mode

```
      IF ( PY( 2 ) .LT. 0.05 ) THEN
         PY( 1 ) = 0.85
         PY( 2 ) = 0.85
         PY( 3 ) = 0.80
         PY( 4 ) = 0.85
         PX( 1 ) = PX( 1 ) + 0.06
         PX( 2 ) = PX( 2 ) + 0.06
         PX( 3 ) = PX( 3 ) + 0.06
         PX( 4 ) = PX( 4 ) + 0.06
      ENDIF

      ENDDO
C     Turn off the event prompt.
❹     CALL GKS$SET_CHOICE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The call to GKS$SET_CHOICE_MODE sets the input operating mode to event. At this point in the program, the choice prompt appears on the workstation surface and the user can trigger the device to place an event report on the input queue.

❷ If there is an event report on the input queue, GKS$AWAIT_EVENT removes the report and GKS$GET_CHOICE retrieves the input values. Notice that the user must trigger the device (or the time specified in an argument to GKS$AWAIT_EVENT must expire) to place an event report on the input queue.

❸ This code draws triangles in columns.

❹ The call to GKS$SET_CHOICE_MODE returns the logical input device to request mode. At this point, the device handler removes the choice prompt from the workstation surface and the user can no longer enter input.

The images generated by this program are identical to the images generated by Example 8–11. The difference is that the user must trigger the device (or allow the time-out argument for GKS$AWAIT_EVENT to expire) before an event report appears on the input queue. Using sample mode, the application program controls the acceptance of the choice without user action.

GET LOCATOR

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$GET_LOCATOR obtains information from the current event report entry in the DEC GKS state list and writes the normalization transformation and the X and Y world coordinate point values to the output arguments.

Syntax

GKS$GET_LOCATOR *(transformation_number, world_x, world_y)*

GGTLC *(x_form, pos_x, pos_y)*

ggetloc *(response)*

Arguments

transformation_number

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This is the argument to which DEC GKS writes the normalization transformation number used to translate the current event's input point to a world coordinate point.

world_x
world_y

data type:	**real**
access:	**write-only**
mechanism:	**by reference**

These are the arguments to which DEC GKS writes the X and Y world coordinate values for the current event record.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
150	GKS$_ERROR_150	No input value of the correct class is in the current event report in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–3.

GET PICK

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$GET_PICK obtains information from the current event report entry in the DEC GKS state list and writes the input status, segment name, and pick identifier to the output arguments.

Syntax

GKS$GET_PICK *(input_status, segment_name, pick_id)*

GGTPK *(input_status, segment_name, pick_id)*

ggetpick *(response)*

Arguments

input_status

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the status of the input values in the current event report. This argument can be either of the following values or constants:

Value	Constant	Description
1	GKS$K_STATUS_OK	Input obtained.
2	GKS$K_STATUS_NOPICK	Triggered without picking.

segment_name

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer representing the segment name in the current event report.

pick_id

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the integer representing the pick identifier for the current event report.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
150	GKS$_ERROR_150	No input value of the correct class is in the current event report in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–4.

GET STRING

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$GET_STRING obtains information from the current event report entry in the DEC GKS state list and writes the string, the string size, and the number of characters written to the output arguments.

When activating string input, the following two buffers exist:

- The application's string buffer, whose size you specify when you pass the buffer argument by descriptor to GKS$GET_STRING.
- The logical input device's string buffer, whose size you can specify in the call to GKS$INIT_STRING.

When reading a string from the current event report, DEC GKS removes characters up to the number that fit into the application's buffer. If the size of the string in the current event report is larger than the application's buffer, you need to call GKS$GET_STRING again, using a larger application buffer, in order to obtain the entire string contained in the report. (Remember that the string contained in the current report does not change until you call GKS$AWAIT_EVENT to replace the current report.)

NOTE

The initial string only appears in the first generated string event report. Subsequent string reports do not contain the initial string.

Syntax

GKS$GET_STRING *(string_buffer, string_size, report_string_size)*

GGTST *(num_char, cstring)*

GGTST- Subset *(num_char, cstring)*

ggetstring *(response)*

Arguments

string_buffer

data type:	**string**
access:	**write-only**
mechanism:	**by descriptor**

This is the argument to which DEC GKS writes the input character string. This is the *application's* buffer.

string_size

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This is the argument to which DEC GKS writes the number of bytes in the string accepted from the current event.

report_string_size

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This is the argument to which DEC GKS writes the total size of the string found in the current event report. If this value is larger than string_size, you may want to call GKS$GET_STRING, passing a larger buffer, to obtain the entire string contained in the report.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
150	GKS$_ERROR_150	No input value of the correct class is in the current event report in routine ****

Program Example

Example 8–17 illustrates the use of the function GKS$GET_STRING.

Example 8–17: Using a String Logical Input Device in Event Mode

```
C     This program initializes and accepts string events from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, DATA_RECORD( 2 ),
     * PROMPT_ECHO_TYPE, ERROR_STATUS, BUFFER_LENGTH,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE, INPUT_STATUS, DEVICE_NUM, STRING_SIZE,
     * CUR_POSITION, REPORT_STRING_SIZE, INCR, INCR2, CLASS
      REAL ECHO_AREA( 4 ), START_X, START_Y, X_VECTOR, Y_VECTOR,
     * LARGER, TIME
      CHARACTER*31 INITIAL_STRING, STRING_BUFFER
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, LARGER / 0.03 /,
     * TIME / 20.0 /

C     First element in the data record is length of the buffer that
C     contains the input string.
      EQUIVALENCE( DATA_RECORD( 1 ), BUFFER_LENGTH )
      EQUIVALENCE( DATA_RECORD( 2 ), CUR_POSITION )
```

(continued on next page)

Example 8–17 (Cont.): Using a String Logical Input Device in Event Mode

```
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      RECORD_BUFFER_LENGTH = 8
      CALL GKS$INQ_STRING_STATE( WS_ID, DEVICE_NUM,
     * ERROR_STATUS, INPUT_MODE, ECHO_FLAG, INITIAL_STRING,
     * STRING_SIZE, PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

❶     ECHO_AREA( 1 ) = 437
      BUFFER_LENGTH = 31
      CUR_POSITION = 1

C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_STRING( WS_ID, DEVICE_NUM, ' ',
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )
```

(continued on next page)

Example 8–17 (Cont.): Using a String Logical Input Device in Event Mode

```
C     Tell the user what is happening.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.05, 0.95, 'Every 20 seconds, type a string.')
      CALL GKS$TEXT( 0.05, 0.90, 'I will use them in my design.')

❷     CALL GKS$SET_STRING_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_EVENT, GKS$K_ECHO )

C     Set the initial text settings and attributes.
      START_X = 0.05
      START_Y = 0.70
      X_VECTOR = 0.0
      Y_VECTOR = 1.0

C     Do for three lines.
      DO 200 INCR = 1, 3, 1

      IF ( INCR .NE. 1 ) THEN

C     Ask for a string...
      CALL GKS$SET_TEXT_UPVEC( 0.0, 1.0 )
      CALL GKS$TEXT( 0.05, 0.85,
     *     'Please enter a string.')
      CALL GKS$SET_TEXT_UPVEC( X_VECTOR, Y_VECTOR )

C     Check the event queue.
      CALL GKS$AWAIT_EVENT( TIME, WS_ID, CLASS, DEVICE_NUM )

C     If the user entered a string, get it...
❸     IF ( CLASS .NE. GKS$K_INPUT_CLASS_NONE ) THEN
         CALL GKS$GET_STRING( STRING_BUFFER, STRING_SIZE,
     *        REPORT_STRING_SIZE )

C     Otherwise, ask for a string...
      ENDIF

      ELSE
C        Provide the first string.
         STRING_BUFFER = 'I''ll give you the first string.'
      ENDIF
```

(continued on next page)

Example 8–17 (Cont.): Using a String Logical Input Device in Event Mode

```
C     Create a design with a text string.
❹     DO 300 INCR2 = 1, 3, 1
         CALL GKS$SET_TEXT_UPVEC( X_VECTOR, Y_VECTOR )
         CALL GKS$TEXT( START_X, START_Y,
     *        STRING_BUFFER )
         X_VECTOR = X_VECTOR + 0.1
         Y_VECTOR = Y_VECTOR - 0.1
300   CONTINUE

C     Reset variables.
      START_Y = START_Y - 0.01
      STRING_BUFFER = ' '

200   CONTINUE

C     Turn off the event prompt.
❺     CALL GKS$SET_STRING_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ Changing the value of variable ECHO_AREA(1) widens the echo area so that the user can enter a larger string than the one which the default area allows.

❷ The call to GKS$SET_STRING_MODE sets the input operating mode to event. At this point in the program, the string prompt appears on the workstation surface and the user can trigger the device to enter an event report on the input queue.

❸ If there is an event report on the input queue, GKS$AWAIT_EVENT removes the report and GKS$GET_STRING retrieves the input values. Notice that the user must trigger the device (or the time specified in an argument to GKS$AWAIT_EVENT must expire) to place an event report on the input queue.

❹ This code outputs the strings by adjusting the character-up vector. This creates a "pinwheel" design on the workstation surface.

❺ The call to GKS$SET_STRING_MODE returns the logical input device to request mode. At this point, the device handler removes the string prompt from the workstation surface and the user can no longer enter input.

The images generated by this program are identical to the images generated by Example 8–13. The difference is that the user must trigger the device (or allow the time-out argument for GKS$AWAIT_EVENT to expire) before an event report appears on the input queue. Using sample mode, the application program controls the acceptance of the string without user action (triggering).

GET STROKE

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$GET_STROKE obtains information from the current event report entry in the DEC GKS state list and writes the normalization transformation, the number of entered points, the stroke point values, and the number of accepted stroke point values to the output arguments.

When activating stroke input, the following two buffers exist:

- The application's stroke buffer, whose size you specify when you pass the buffer argument by descriptor to GKS$GET_STROKE.
- The logical input device's stroke buffer, whose size you can specify in the call to GKS$INIT_STROKE.

When reading stroke points from the current event report, DEC GKS removes points up to the number that fit into the application's buffer. If the size of the stroke in the current event report is larger than the application's buffer, you need to call GKS$GET_STROKE again, using a larger application buffer, in order to obtain the entire stroke contained in the report. (Remember that the stroke contained in the current report does not change until you call GKS$AWAIT_EVENT to replace the current report.)

NOTE

The initial stroke appears only in the first generated stroke event report. Subsequent stroke reports do not contain the initial stroke.

Syntax

GKS$GET_STROKE *(transformation_number, num_entered_points, stroke_buffer_x, stroke_buffer_y, stroke_size_x, stroke_size_y)*

GSTSK *(max_pts, xform, num_pts, px, py)*

gsetstroke *(response)*

Arguments

transformation_number

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the number of the normalization transformation used to translate points in the current event report to world coordinate points.

num_entered_points

data type: **integer**
access: **write-only**
mechanism: **by reference**

This is the argument to which DEC GKS writes the number of points in the stroke in the current event report.

stroke_buffer_x
stroke_buffer_y

data type: **array (real)**
access: **write-only**
mechanism: **by descriptor**

These are the arguments to which DEC GKS writes the X and Y world coordinate values from the current event report. These arguments are the application's stroke buffer.

stroke_size_x
stroke_size_y

data type: **integer**
access: **write-only**
mechanism: **by reference**

These are the arguments to which DEC GKS writes the number of stroke points actually accepted from the current event report and placed in the application buffer. If the values in these arguments are less than num_entered_points, then you may want to call GKS$GET_STROKE again, passing a larger bufffer, to obtain the entire stroke entered.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
150	GKS$_ERROR_150	No input value of the correct class is in the current event report in routine ****

Program Example

Example 8–18 illustrates the use of the function GKS$GET_STROKE.

Example 8–18: Using a Stroke Logical Input Device in Event Mode

```
C     This program initializes and accepts stroke events from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, DATA_RECORD( 6 ), BUFFER_SIZE,
     * DIMENSION, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * TRANSFRM, NUM_ENTERED_POINTS, INPUT_MODE, ECHO_FLAG,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE, INPUT_STATUS, DEVICE_NUM,
     * RET_SIZE_X, RET_SIZE_Y, I, EDIT_POSITION, ATTS_FLAG, TEXT,
     * INCR, INCR2, RET_SIZE_BUF( 3 ), CLASS
      REAL ECHO_AREA( 4 ), STROKE_X( 50 ),
     * STROKE_Y( 50 ), X_INT, Y_INT, TIME_INT,
     * STROKE_BUFFER_X( 3, 50 ), STROKE_BUFFER_Y( 3, 50 ),
     * LARGER, TIME
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, TEXT / 1 /,
     * LARGER / 0.03 /, TIME / 20.0 /

C     First element in the data record is the buffer size.
      EQUIVALENCE( DATA_RECORD( 1 ), BUFFER_SIZE)
      EQUIVALENCE( DATA_RECORD( 2 ), EDIT_POSITION)
      EQUIVALENCE( DATA_RECORD( 3 ), X_INT)
      EQUIVALENCE( DATA_RECORD( 4 ), Y_INT)
      EQUIVALENCE( DATA_RECORD( 5 ), TIME_INT)
      EQUIVALENCE( DATA_RECORD( 6 ), ATTS_FLAG)

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      RECORD_BUFFER_LENGTH = 24
      CALL GKS$INQ_STROKE_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, DIMENSION, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, TRANSFRM, NUM_ENTERED_POINTS,
     * STROKE_X, STROKE_Y, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH, RECORD_SIZE )

C     Allow a buffer that is large enough.
      BUFFER_SIZE = 256

C     By specifying to DEC GKS to use the current attributes flag, you
C     need to pass the 24 byte data record instead of the 52 byte record.
      ATTS_FLAG = GKS$K_ACF_CURRENT
```

(continued on next page)

Example 8–18 (Cont.): Using a Stroke Logical Input Device in Event Mode

```
C     To initialize a device, make sure it's in request mode (the DEC
C     GKS default).
      CALL GKS$SET_LOCATOR_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

      CALL GKS$INIT_STROKE( WS_ID, DEVICE_NUM,
     * NUM_ENTERED_POINTS, STROKE_X, STROKE_Y, TRANSFRM,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )

(1)   CALL GKS$SET_STROKE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_EVENT, GKS$K_ECHO )

C     Tell the user how many sets of stroke points to enter.
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( 0.05, 0.95,
     * 'Every 20 seconds, enter points.')
      CALL GKS$TEXT( 0.05, 0.90,
     * 'I''ll show the three fill areas.')

C     Do for three sets of stroke points.
      DO 200 INCR = 1, 3, 1

C     Check the event queue.
      CALL GKS$AWAIT_EVENT( TIME, WS_ID, CLASS, DEVICE_NUM )

C     If the user entered a stroke, get it...
(2)   IF ( CLASS .NE. GKS$K_INPUT_CLASS_NONE ) THEN
         CALL GKS$GET_STROKE( TRANSFRM, NUM_ENTERED_POINTS,
     *        %DESCR( STROKE_X ), %DESCR( STROKE_Y ),
     *        RET_SIZE_X, RET_SIZE_Y )
      ENDIF

      RET_SIZE_BUF( INCR ) = MIN( RET_SIZE_X, RET_SIZE_Y )
```

(continued on next page)

Example 8–18 (Cont.): Using a Stroke Logical Input Device in Event Mode

```
C     Put the strokes in a buffer.
      DO 300 INCR2 = 1, RET_SIZE_BUF( INCR ), 1
      STROKE_BUFFER_X( INCR, INCR2 ) = STROKE_X( INCR2 )
      STROKE_BUFFER_Y( INCR, INCR2 ) = STROKE_Y( INCR2 )
300   CONTINUE

200   CONTINUE

C     Get rid of the stroke prompt...
      CALL GKS$SET_STROKE_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

C     Present the corresponding fill areas.  Press RETURN when you are
C     ready to view the next fill area.
      CALL GKS$CLEAR_WS( WS_ID, GKS$K_CLEAR_ALWAYS )
      CALL GKS$TEXT( 0.05, 0.95,
     * 'Here are the fill areas.')
      CALL GKS$CREATE_SEG( TEXT )
      CALL GKS$TEXT( 0.05, 0.90,
     * 'Press RETURN when ready.')
      CALL GKS$CLOSE_SEG()

      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )

      DO 400 INCR = 1, 3, 1

C     Put the current stroke in the temporary buffer.
      DO 500 INCR2 = 1, RET_SIZE_BUF( INCR ), 1
      STROKE_X( INCR2 ) = STROKE_BUFFER_X( INCR, INCR2 )
      STROKE_Y( INCR2 ) = STROKE_BUFFER_Y( INCR, INCR2 )
500   CONTINUE

      CALL GKS$FILL_AREA( RET_SIZE_BUF( INCR ),
     * STROKE_X, STROKE_Y )
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
      IF ( INCR .EQ. 3 ) THEN
         CALL GKS$DELETE_SEG( TEXT )
      ENDIF
      CALL GKS$REDRAW_SEG_ON_WS( WS_ID )

400   CONTINUE
```

(continued on next page)

Example 8–18 (Cont.): Using a Stroke Logical Input Device in Event Mode

```
CALL GKS$DEACTIVATE_WS( WS_ID )
CALL GKS$CLOSE_WS( WS_ID )
CALL GKS$CLOSE_GKS()
END
```

The following numbers correspond to the numbers in the previous example:

❶ The call to GKS$SET_STROKE_MODE sets the input operating mode to event. At this point in the program, the stroke prompt appears on the workstation surface and the user can enter stroke points, trigger the device, and enter the event report on the input queue.

❷ If there is an event report on the input queue, GKS$AWAIT_EVENT removes the report and GKS$GET_STROKE retrieves the input values. Notice that the user must trigger the device (or the time specified in an argument to GKS$AWAIT_EVENT must expire) to place an event report on the input queue.

❸ This code sets the correct size of the buffer and stores the stroke points in a two-dimensional array.

❹ The call to GKS$SET_STROKE_MODE returns the logical input device to request mode. At this point, the device handler removes the stroke prompt from the workstation surface and the user can no longer enter input.

❺ This code uses each of the sets of stroke points to create and display a fill area.

The images generated by this program are identical to the images generated by Example 8–14. The difference is that the user must trigger the device (or allow the time-out argument for GKS$AWAIT_EVENT to expire) before an event report appears on the input queue. Using sample mode, the application program controls the acceptance of the stroke without user action (triggering).

GET VALUATOR

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$GET_VALUATOR obtains information from the current event report entry in the DEC GKS state list and writes the real value output argument.

Syntax

GKS$GET_VALUATOR *(real_value)*

GGTVL *(value)*

ggetval *(response)*

Arguments

real_value

data type:	**real**
access:	**write-only**
mechanism:	**by reference**

This is the argument to which DEC GKS writes the current measure of the valuator device.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state; GKS shall be in one of the states WSOP, WSAC, or SGOP
150	GKS$_ERROR_150	No input value of the correct class is in the current event report in routine ****

Program Example

To see an example of a call to this function, refer to Example 8–4.

Chapter 9

Segment Functions

The DEC GKS segment functions create, manipulate, and delete stored groups of output primitives called *segments*. The segment functions can be divided into the following categories:

Category	GKS Functions
Using Segments	GKS$ASSOC_SEG_WITH_WS, GKS$CLOSE_SEG, GKS$COPY_SEG_TO_WS, GKS$CREATE_SEG, GKS$DELETE_SEG, GKS$DELETE_SEG_FROM_WS, GKS$INSERT_SEG, GKS$RENAME_SEG
Primitive Attributes	GKS$SET_PICK_ID
Segment Attributes	GKS$SET_SEG_DETECTABILITY, GKS$SET_SEG_HIGHLIGHTING, GKS$SET_SEG_PRIORITY, GKS$SET_SEG_VISIBILITY
Segment Transformations	GKS$ACCUM_XFORM_MATRIX, GKS$EVAL_XFORM_MATRIX, GKS$SET_SEG_XFORM

When producing output, you may wish to reproduce a graphic image at different positions within a single picture, possibly across different devices, and possibly at different points during program execution. It is inefficient to call all of the DEC GKS output and attribute functions every time you want to reproduce such an image. DEC GKS provides a method of storing groups of output primitives, output attributes, and clipping information in a segment.

9.1 Creating, Using, and Deleting Segments

To use segments, your workstation should be one of the categories GKSK_WSCAT_OUTPUT, GKSK_WSCAT_OUTIN, GKS$K_WSCAT_MO, or GKS$K_WSCAT_WISS (described in Section 9.2). When you create a segment, the segment is stored on all active workstations.

To create a segment, DEC GKS must be in the operating state GKS$K_WSAC (at least one workstation active). When DEC GKS is in this state, you can call GKS$CREATE_SEG, which creates a segment on all active workstations. The only argument passed to GKS$CREATE_SEG is the *segment name*. You use a segment name to identify a particular segment.

After you call the function GKS$CREATE_SEG, the DEC GKS operating state changes to GKS$K_SGOP (segment open). Subsequent calls to the DEC GKS output and output attribute functions produce primitives that are stored in the segment on all active workstations. When you have created the desired image, call the function GKS$CLOSE_SEG. This call closes the segment, causing the DEC GKS operating state to change back to GKS$K_WSAC.

For a clearer understanding of the segment creation process, review the following code example:

```
      .
      .
      .
INTEGER BOX, NUM_POINTS, WS_ID
REAL X_VALUES( 5 ), Y_VALUES ( 5 )
DATA X_VALUES / 0.0, 1.0, 1.0, 0.0, 0.0 /
DATA Y_VALUES / 0.0, 0.0, 1.0, 1.0, 0.0 /
DATA WS_ID / 1 /, BOX / 1 /, NUM_POINTS / 5 /
      .
      .
      .
CALL GKS$ACTIVATE_WS( WS_ID )
      .
      .
      .
CALL GKS$CREATE_SEG( BOX )
CALL GKS$POLYLINE( NUM_POINTS, X_VALUES, Y_VALUES )
CALL GKS$CLOSE_SEG()
      .
      .
      .
```

In this code example, the workstation WS_ID stores the segment BOX that contains an outline of a box.

When you call the function GKS$CREATE_SEG, the DEC GKS operating state changes from GKS$K_WSAC to GKS$K_SGOP. GKS$K_SGOP signifies that a segment is *open*, or being created. Also, calling GKS$CREATE_SEG establishes the segment state list associated with the segment name, and DEC GKS records the segment name (in the DEC GKS state list) as the name of the currently open segment.

Segments cannot contain other segments, or in other words, segments cannot be nested. Therefore, if you call GKS$CREATE_SEG, you must call GKS$CLOSE_SEG before you attempt to call GKS$CREATE_SEG again. Until you call GKS$CLOSE_SEG, DEC GKS associates all generated output primitives with the name of the open segment. When you call GKS$CLOSE_SEG, the DEC GKS operating state changes from GKS$K_SGOP back to GKS$K_WSAC. After you close a segment, you cannot reopen the segment to add more output primitives.

If you need to, you can rename the segment using the function GKS$RENAME_SEG. If you are keeping an ordered list of segments, calls to this function may be useful.

There are three ways to delete segments. If you use the function GKS$DELETE_SEG_FROM_WS, DEC GKS deletes the segment from the specified workstation. If you use GKS$DELETE_SEG, DEC GKS deletes the specified segment from all workstations storing the segment. If you call GKS$CLEAR_WS, and if the surface is cleared, you delete all segments stored on that workstation.

For more information concerning the DEC GKS operating states or the segment state list, refer to Chapter 4, Control Functions.

NOTE

If you store primitives in a segment and if you want to be able to change the primitive's appearance elsewhere in the program, you must set the primitive's ASF to be GKS$K_ASF_BUNDLED before you generate the primitive. In this way, the primitive's ASF is stored in the segment with the primitive. If you want to change the primitive's appearance, you call the appropriate SET REPRESENTATION function for the primitive's bundle index. If you store the primitive in a segment using individual attributes, the appearance of the primitive cannot be changed after primitive generation. For more information, refer to Chapter 6, Output Attribute Functions.

9.1.1 Pick Identification

One of the DEC GKS logical input classes is the *pick* input class. Through this process, the user can choose a segment, and possibly a portion of the segment, as displayed on the surface of the workstation.

The following is an example of a call requesting pick input:

```
      .
      .
      .
 CALL GKS$REQUEST_PICK( 1, ARG_1, ARG_2,
* SEGMENT_NAME, PICK_ID )
      .
      .
      .
```

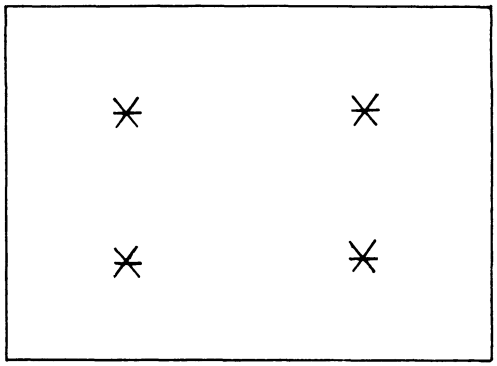
The arguments SEGMENT_NAME and PICK_ID are write-only arguments. At the completion of the function call, SEGMENT_NAME contains the name value of the chosen segment.

Also, after completion of the function call, the argument PICK_ID contains a value called the *pick identifier* of the primitive chosen during pick input. The pick identifier is a numeric output attribute. Like other output attribute values (line type, line width, color, text alignment, and so forth), the pick identifier is bound to an output primitive at the time of generation, and you cannot change its value. However, you can change the current pick identifier value before generating each output primitive. In doing so, DEC GKS associates a different numeric pick identifier value with each generated primitive.

During segment creation, you can use pick identifiers to establish a hierarchy within the segment. During pick input, DEC GKS returns the same segment name if the pick prompt touches the same segment, but may return different pick identifiers depending on which primitive *within the segment* the pick prompt touches.

For example, Figure 9–1 shows five distinct output primitives (one line and four markers). The following example shows how to assign different pick identifiers to distinct primitives within the same segment.

Figure 9–1: Primitives Within a Segment



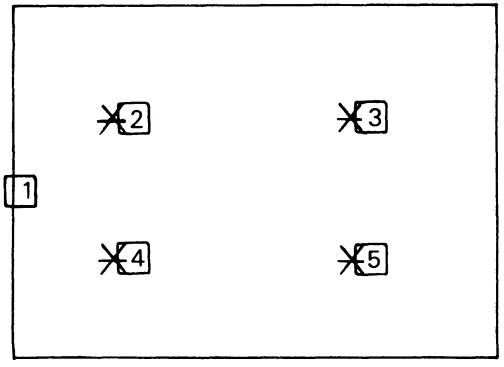
Each of the five primitives within this segment can have a different pick identifier.

ZK-5213-86

```
          .
          .
          .
C     The segment's name is 99.
      CALL GKS$CREATE_SEG( 99 )

C     The line's pick identifier is 1.
      CALL GKS$SET_PICK_ID( 1 )
      CALL GKS$POLYLINE( 5, X_VALUES, Y_VALUES )

C     The marker in the upper left corner has a pick identifier of 2.
      CALL GKS$SET_PICK_ID( 2 )
      CALL GKS$POLYMARKER( 1, X_VALUE_1, Y_VALUE_1 )

C     The marker in the upper right corner has a pick identifier of 3.
      CALL GKS$SET_PICK_ID( 3 )
      CALL GKS$POLYMARKER( 1, X_VALUE_2, Y_VALUE_2 )

C     The marker in the lower left corner has a pick identifier of 4.
      CALL GKS$SET_PICK_ID( 4 )
      CALL GKS$POLYMARKER( 1, X_VALUE_3, Y_VALUE_3 )

C     The marker in the lower right corner has a pick identifier of 5.
      CALL GKS$SET_PICK_ID( 5 )
      CALL GKS$POLYMARKER( 1, X_VALUE_4, Y_VALUE_4 )

      CALL GKS$CLOSE_SEG()
          .
          .
          .
```

Figure 9–2 illustrates the pick identifiers returned depending on the position of the pick prompt during input. To see how to use pick identifiers, refer to the program example for GKS$SET_PICK_ID in this chapter.

Figure 9–2: Returned Pick Identifiers



Given the position of each of the prompts, the graphics handler returns the pick identifier value listed in each prompt. For all prompts, the graphics handler returns the segment name 99.

ZK-5214-86

9.2 Workstations and Segment Storage

When DEC GKS stores a segment on an active GKSK_WSCAT_OUTPUT, GKSK_WSCAT_OUTIN, or GKS$K_WSCAT_MO workstation, the method of storage is called workstation dependent segment storage (WDSS). On these workstations, you can control the segment attributes (see Section 9.4), you can move or alter the shape of the segment using the segment transformation functions (see Section 9.4.4.1), or you can delete the segment (either from a single workstation or from all workstations storing the segment).

If you are creating segments using the WDSS method of storage, you *cannot* copy a segment from one workstation to another. Also, you cannot recall a segment once it has been deleted from a workstation. You can only alter the segment's position within the picture by changing the segment transformation.

To copy a segment, or to reassociate a segment with a workstation after deletion from that particular workstation, you need to store the segment in workstation independent segment storage (WISS). Once a segment is stored in WISS, the segment is independent of any workstation and can be copied from WISS to other workstations.

By storing a segment on a WISS workstation, you can delete a segment from a particular workstation (that is not WISS). Then, when you need to use the deleted segment later in the program, you can *associate* the segment stored on WISS with the other workstation, *copy* the segment to the other workstation, or *insert* the segment's primitives into the output stream of the other workstation.

If you associate a segment stored on a WISS workstation with another workstation, the other workstation stores an identical segment. If you copy a segment from a WISS workstation to another workstation, the segment's primitives are copied to the surface of the second workstation, but the second workstation does *not* store them in a segment. If you insert a segment into the output stream of another workstation, DEC GKS transforms and then copies all of the segment's primitives onto the surface of the other workstation, but the second workstation does *not* store them in a segment. If you are creating a segment, you can insert another segment's primitives into the newly created segment, but those primitives become part of the new segment and are no longer bound by the old segment name (see GKS$INSERT_SEG in this chapter for more information).

DEC GKS implements the WISS data structure as a workstation. To store a segment using WISS, open and then activate WISS specifying GKS$K_WSTYPE_WISS (value 5) as the workstation type. When you open WISS, you can specify GKS$K_CONID_DEFAULT as the connection identifier argument. (If you specify GKS$K_WSTYPE_WISS, DEC GKS ignores the connection identifier argument).

Once you activate the WISS workstation and create segments, you can use the DEC GKS functions GKS$ASSOC_SEG_WITH_WS, GKS$COPY_SEG_TO_WS, and GKS$INSERT_SEG. Example 9–1 shows the difference between GKS$ASSOC_SEG_WITH_WS and GKS$COPY_SEG_TO_WS. To see a program example using GKS$INSERT_SEG, refer to the appropriate function description in this chapter.

Example 9–1: Comparing GKS$ASSOC_SEG_WITH_WS and GKS$COPY_SEG_TO_WS

```
C     This program draws a house in the lower left corner of the
C     screen and a line of text in the upper left corner. The program
C     redraws the segments.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE,
     * UPPER_LEFT_CORNER, WISS, TITLE, NUM_POINTS
      REAL PX ( 9 ), PY ( 9 ), WORLD_X, WORLD_Y, LARGER
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE / 1 /, UPPER_LEFT_CORNER / 2 /, WISS / 2 /,
     * TITLE / 2 /, WORLD_X / 0.1 /, WORLD_Y / 0.5 /,
     * LARGER / 0.03 /, WS_ID/ 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$OPEN_WS( WISS, GKS$K_CONID_DEFAULT,
     * GKS$K_WSTYPE_WISS )

C     Only activate the WISS workstation.
(1)   CALL GKS$ACTIVATE_WS( WISS )

C     Create two segments and store them on the WISS workstation.
      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
      CALL GKS$SET_VIEWPORT( UPPER_LEFT_CORNER, 0.0, 0.5, 0.5, 1.0 )

      CALL GKS$CREATE_SEG( HOUSE )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

      CALL GKS$CREATE_SEG( TITLE )
      CALL GKS$SELECT_XFORM( UPPER_LEFT_CORNER )
      CALL GKS$SET_TEXT_HEIGHT( LARGER )
      CALL GKS$TEXT( WORLD_X, WORLD_Y, 'Associated segment.' )
      CALL GKS$CLOSE_SEG()
```

(continued on next page)

Example 9–1 (Cont.): Comparing GKS$ASSOC_SEG_WITH_WS and GKS$COPY_SEG_TO_WS

```
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Associate the text with the VT241, but only copy the house.
❷     CALL GKS$ASSOC_SEG_WITH_WS( WS_ID, TITLE )
      CALL GKS$COPY_SEG_TO_WS( WS_ID, HOUSE )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Redraw all segments forcing an update to the screen.
❸     CALL GKS$REDRAW_SEG_ON_WS( WS_ID )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$DEACTIVATE_WS( WISS )
      CALL GKS$CLOSE_WS( WISS )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

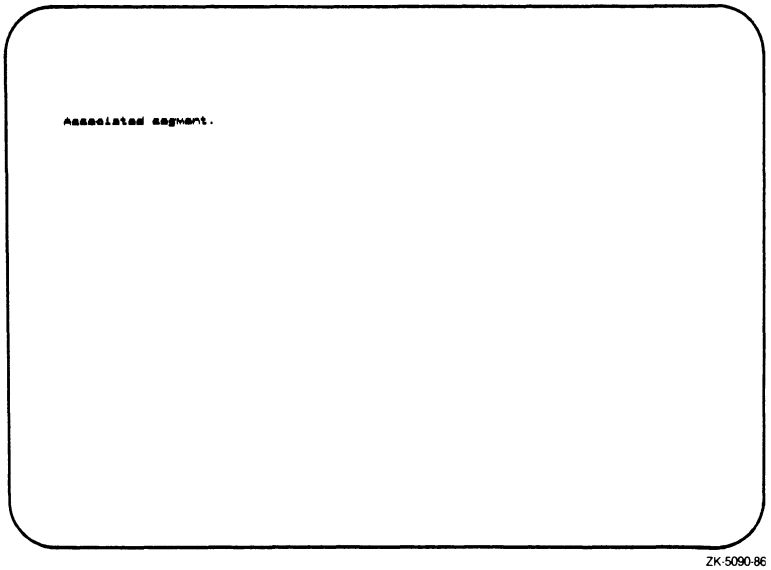
❶ If you activate only the WISS workstation, DEC GKS stores created segments only on the WISS workstation. The VT241 screen remains clear.

❷ By associating the segment containing text, the VT241 stores the segment. By copying the segment containing the house, DEC GKS draws the primitives to the VT241 screen, but the VT241 does not store the segment.

❸ When you redraw all segments, you force DEC GKS to update the screen, eliminating all primitives not contained in segments. The house disappears and the text remains on the VT241 since it is stored as a segment.

Figure 9–3 shows the screen of a VT241 terminal after the program has run to completion.

Figure 9–3: Comparing GKS$ASSOC_SEG_WITH_WS and GKS$COPY_SEG_TO_WS—VT241



ZK-5090-86

9.3 Segments and Surface Update

When you request changes to segment attributes (described in Section 9.4), the change may take place immediately (dynamically) or DEC GKS may need to update the surface to implement the change (an implicit regeneration), depending on the capabilities of your device. An implicit regeneration clears the screen and only redraws the primitives stored in segments. All segments not stored in primitives are lost. You can use the function GKS$INQ_DYN_MOD_SEG_ATTB to determine if a request for a segment attribute change requires an implicit regeneration on your device.

There are two ways to determine whether your device requires an implicit regeneration to implement a change. If you are making only a few changes, you can call GKS$INQ_WS_DEFER_AND_UPDATE to determine if the *new frame necessary at update* flag is YES. If you are making many different changes, calling GKS$INQ_WS_DEFER_AND_UPDATE each time is inefficient.

You can call GKS$INQ_DYN_MOD_SEG_ATTB once to determine for which changes your workstation requires an implicit regeneration. Then, you can set flags to force regenerations only when you make changes that require them. If you need to regenerate the picture on the workstation surface when changing segment attributes, you simply call GKS$UPDATE_WS and pass GKS$K_PERFORM_FLAG as an argument.

NOTE

If you want to redraw all of the segments on the workstation surface regardless of the current status of the *new frame* flag, you can call GKS$REDRAW_SEG_ON_WS. A call to this function is equivalent to a call to GKS$UPDATE_WS while the *new frame* flag is set to YES, and while passing the argument GKS$K_PERFORM_FLAG.

The following requests for changes to segments may require an implicit regeneration of the screen depending on the capabilities of your device (see Section 9.4 for a complete descriptions of the segment attributes):

Change	Possible Effect
Segment priority	If you call the following functions: • GKS$DELETE_SEG • GKS$DELETE_SEG_FROM_WS • GKS$ASSOC_SEG_WITH_WS • GKS$SET_SEG_PRIORITY • GKS$SET_SEG_XFORM • GKS$SET_SEG_VISIBILITY You may have created a situation in which two segments of different priorities overlap, or in which an overlapped segment must now be made completely visible, or in which visibility changes. In all cases, DEC GKS must take the segments' priorities into consideration before determining if the picture is out of date.
Segment transformation	Many workstations are unable to reposition segments dynamically, thus requiring an implicit regeneration.

Change	Possible Effect
Segment visibility	Some workstations may be able to make an invisible segment visible dynamically, but may need an implicit regeneration to make visible segments invisible, since visible-to-invisible changes require that the segments "beneath" the segment be redrawn. Some workstations may need an implicit regeneration to perform both, and some workstations may be able to make both changes dynamically.
Segment highlighting	Some workstations may need to implicitly regenerate the surface before they can highlight a segment.
Segment deletion	Segment deletion may require reproducing the segments "beneath" the deleted segment. Calling either GKS$DELETE_SEG or GKS$DELETE_SEG_FROM_WS can require an implicit regeneration of the screen, depending on the capabilities of your workstation.

There are other conditions under which DEC GKS may require a surface regeneration, depending on the capabilities of your device. For example, if you attempt to alter the polyline representation (refer to Chapter 6, Output Attribute Functions), the VT241 requires an implicit regeneration to affect this change.

Therefore, if you are going to make certain output attribute changes or workstation transformation changes, you need to put all important output primitives into segments so that they are not lost when you update the surface. For complete information as to changes that may require implicit regeneration, on GKS$UPDATE_WS, or on GKS$REDRAW_SEG_ON_WS, refer to Chapter 4, Control Functions.

9.4 Segment Attributes

Just as a workstation stores the output attributes of a primitive when it is a part of a segment, a workstation stores *segment attributes* that affect all of the primitives stored within a segment. The segment attributes are as follows:

- Detectability
- Highlighting
- Priority
- Transformation
- Visibility

The following sections describe the segment attributes in detail.

9.4.1 Detectability

This segment attribute determines whether or not the segment can be chosen during pick input. Pick input is only available on GKS$K_WSCAT_OUTIN workstations. By default, DEC GKS segments are undetectable (GKS$K_UNDETECTABLE).

In order for you to pick a segment, it must be both detectable and visible (GKS$K_VISIBLE). In many applications, if you do not want the user to be able to pick a segment, you should make the segment invisible as well as undetectable. Remember that making a segment undetectable does not make the segment invisible; these are two separate segment attributes.

For more information concerning detectability, refer to GKS$SET_SEG_DETECTABILITY in this chapter. For more information concerning pick input, refer to Chapter 8, Input Functions.

9.4.2 Highlighting

This segment attribute determines whether or not a workstation presents a highlighted segment on the workstation surface to draw the attention of the user to that segment. By default, DEC GKS segments are not highlighted (GKS$K_NORMAL).

Highlighting is device dependent and can be implemented in any of the following ways:

- Blinking all primitives in a segment
- Outlining the *segment extent rectangle*
- Reversing the foreground and background colors within the segment extent rectangle
- Outlining of all output primitives stored within the segment

The segment extent rectangle is the rectangle that outlines all of the NDC points of the primitives stored in the segment. For more information concerning highlighting, refer to GKS$SET_SEG_HIGHLIGHTING in this chapter.

9.4.3 Priority

This segment attribute determines which segment's primitives take priority when two segments overlap on the workstation surface. To assign a priority to a segment, you assign to the segment a real number greater than or equal to the value 0.0, and less than or equal to the value 1.0. Segments with the priority 0.0 have the lowest priority, and segments with the priority 1.0 have the highest priority. By default, DEC GKS segments have a priority value of 0.0.

Different devices implement segment priority differently. Either a device supports an infinite number of priorities (theoretically), or the device supports a specific number of priorities. If the device supports an infinite amount of priorities, the *maximum number of segment priorities supported* entry in the workstation description table is the value 0. Otherwise, the entry contains the number of priorities supported. (To access this table entry, call the function GKS$INQ_SEG_PRIORITY.)

If the number of priorities supported is not 0 (specifying an infinite number of supported priorities), then DEC GKS divides the 0.0 to 1.0 priority range into subranges according to the number of supported priorities. If you specify, for two different segments, two different priority values that fall within the same subrange, those segments have the same priority. For instance, if a workstation supports two segment priorities, all segments with the specified values between 0.0 and 0.5 inclusive have the same priority, and values between 0.51 and 1.0 have the same priority.

For more information concerning segment priority, refer to GKS$SET_SEG_PRIORITY in this chapter.

9.4.4 Transformation

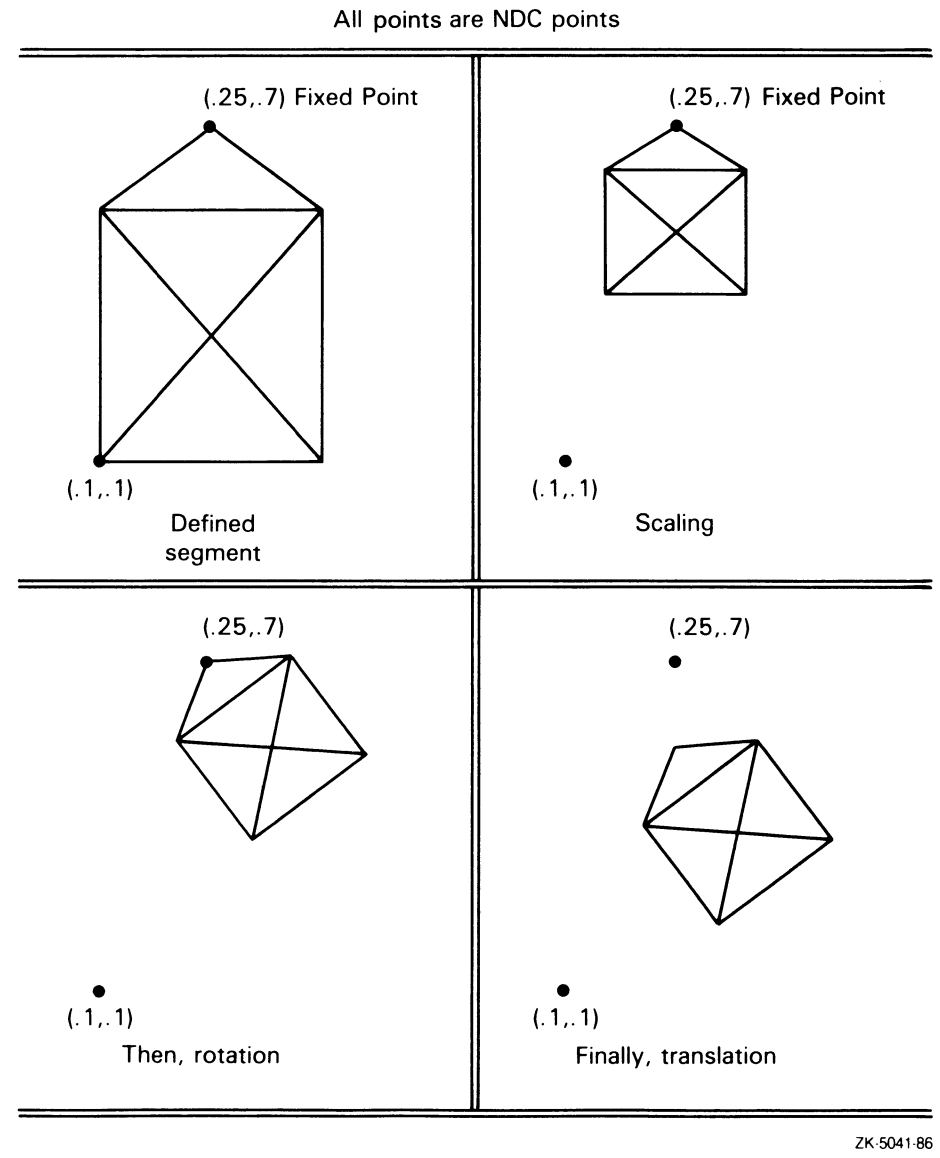
When DEC GKS creates a picture containing segments, it places into effect the current normalization transformation, applies the current *segment transformation* to each segment, and if you have enabled clipping, clips the picture at the current normalization viewport. By default, DEC GKS applies the *identity segment transformation* to all segments. The identity transformation makes no changes to the size or position of the segment.

If you desire, you can change the segment transformation that affects the following components of segment appearance.

Component	Description
Scaling	The first step in the segment transformation process is to scale the segment. Scaling determines the size of the segment extent rectangle, either enlarging or decreasing the total size of the segment.
Rotation	The second step in the segment transformation process is to rotate the segment. Rotation determines the positioning of the segment by establishing a fixed coordinate point in the segment, and then rotating the remaining segment points around the fixed point axis by a specified number of radians.
Translation	The last step in the segment transformation process is to translate the segment's coordinate points to new points according to vector coordinate values. Simply, it shifts the segment position in NDC space.

Figure 9–4 illustrates the effects of scaling, rotation, and shifting.

Figure 9–4: Scaling, Rotation, and Translation



The first decision you must make when working with segment transformations is whether to specify your fixed point in a world or NDC coordinate point. If you want to transform portions of the segment according to the *current* normalization transformation mapping, specify world coordinates. DEC GKS maps the specified world coordinate point to the NDC plane and then performs the rotation or scaling.

If you want to transform the segment as stored on the NDC space (regardless of the current normalization transformation), specify an NDC point as your fixed point.

Next, if you want to scale or to rotate the segment, you must decide which point in the segment to use as an axis, or a *fixed point.* When DEC GKS scales the segment, the fixed point is the only point that maintains its position as the segment decreases or increases in size either towards or away from the fixed point. When DEC GKS rotates the segment, it uses the fixed point as the axis around which the other points in the segment rotate. If you do not wish to scale or rotate the segment, you can specify the value 0.0 for both fixed point coordinate values.

If you decide to shift the segment, you need to establish a *translation vector.* The translation vector is expressed by two real number values that specify by how much the X and Y segment coordinate values change. When DEC GKS translates the segment, it adds the values specified in the translation vector to the segment's X and Y values, moving the segment within the specified coordinate plane. If you do not wish to translate the segment's position, you can specify the value 0.0 for both components of the translation vector.

If you decide to rotate the segment, you must decide on an *angle of rotation* in radians. A radian is a measure of an angle. A full circle, 360 degrees, equals 2*pi radians, one radian equaling 180/pi degrees. The value pi equals approximately 3.14. DEC GKS rotates the segment on the axis of the fixed point by the radian specified as the angle of rotation. Positive rotation values rotate counter clockwise; negative rotation values rotate clockwise. If you do not wish to rotate the segment, you can specify 0.0 for the angle of rotation.

Finally, if you decide to scale the segment, you need to establish the *scale factors.* You express a scale factor as two real number values; DEC GKS multiplies the X and Y segment coordinate values by the scale factor components to determine the new size of the segment. If you do not want to scale the segment (keeping the segment the same size), specify the value 1.0 for both components of the scale factor. Values less than 1.0 decrease the segment size, and values greater than 1.0 increase the segment size.

Once you have decided how to scale, rotate, and translate a segment, you must construct a *transformation matrix.* A transformation matrix is a six-element array of real values. To assist you in the creation of of a transformation matrix, DEC GKS provides the utility functions GKS$EVAL_XFORM_MATRIX and GKS$ACCUM_XFORM_MATRIX. The function GKS$EVAL_XFORM_MATRIX has the following function syntax:

```
GKS$EVAL_XFORM_MATRIX ( fixed_point_x, fixed_point_y, translation_vec_x,
translation_vec_y, rotation, scale_x, scale_y, type_of_coordinates,
transformation_matrix )
```

After evaluating the first eight arguments, GKS$EVAL_XFORM_MATRIX establishes the appropriate transformation matrix and writes the six-element array of real numbers to the last argument *transformation_matrix.* For detailed information concerning this function, refer to GKS$EVAL_XFORM_MATRIX in this chapter.

The function GKS$ACCUM_XFORM_MATRIX is identical to GKS$EVAL_XFORM_MATRIX, except that its first read-only argument is another transformation matrix, as follows:

```
GKS$ACCUM_XFORM_MATRIX ( first_xform_matrix, fixed_point_x, fixed_point_y,
translation_vec_x, translation_vec_y, rotation, scale_x, scale_y, type_of_coordinates,
accumulated_xform_matrix )
```

If you have a previously constructed transformation matrix to which you want to add translation, shifting, and scaling values, you call GKS$ACCUM_XFORM_MATRIX. DEC GKS creates a new transformation matrix using the first matrix and the specified scaling, rotation, and translation information, and then returns the resulting transformation matrix to the last argument.

Once you have established the desired transformation matrix, either by accumulating matrixes or by evaluating a single matrix, you can set the segment transformation using GKS$SET_SEG_XFORM, which takes the name of a segment and the transformation matrix identifier as its arguments. DEC GKS applies the specified transformation to the stored segment on the NDC plane. This current transformation remains in effect until you change it. Before copying a segment, associating a segment, or inserting a segment on a workstation, DEC GKS first checks the current segment transformation in the segment state list, and applies that transformation to the stored segment.

You may have to update the workstation surface in order to see the change in the segment transformation. See Section 9.3 for more information concerning surface update.

To illustrate the entire process of segment transformation, review the following sequence of examples and figures.

In Example 9–2, all of the coordinate values are world coordinates. The fixed point is the tip of the house (0.25, 0.9). The translation vector represents the number of X and Y values by which we want to increase or decrease the present coordinates; in this example, the Y values increase by two world coordinate point units (0.0, 0.2). The angle of rotation is 30 degrees (pi/6 radians). The scale factors represent a 50 percent decrease in size for both the X and Y axes (0.5, 0.5).

If we specify all of the necessary arguments in a single call to GKS$EVAL_XFORM_MATRIX, DEC GKS always scales, then rotates, and finally translates the segment, in that order. Following the example, Figure 9–5 illustrates the effects of the example on a VT241.

Example 9–2: The Effects of a Segment Transformation

```
C     This program transforms the house contained in a segment.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE, NUM_POINTS
      REAL FIXED_X, FIXED_Y, VECTOR_X, VECTOR_Y, ROTATION,
     * SCALE_X, SCALE_Y, NULL, XFORM_MATRIX( 6 )
      REAL PX ( 9 ), PY ( 9 )
      DATA FIXED_X / 0.25 / FIXED_Y / 0.9 /, VECTOR_X / 0.0 /,
     * VECTOR_Y / 0.2 /, SCALE_X / 0.5 /, SCALE_Y / 0.5 /,
     * NULL / 0.0 /, NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE / 1 /, WS_ID / 1 /
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$SET_CLIPPING( GKS$K_NOCLIP )

      CALL GKS$CREATE_SEG( HOUSE )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Rotation equals pi divided by 6 (30 degrees).
      ROTATION = 3.14/6.0

C     Set scaling, rotation, and translation.
(1)   CALL GKS$EVAL_XFORM_MATRIX( FIXED_X, FIXED_Y, VECTOR_X,
     * VECTOR_Y, ROTATION, SCALE_X, SCALE_Y,
     * GKS$K_COORDINATES_WC, XFORM_MATRIX )

C     Transform the segment.
(2)   CALL GKS$SET_SEG_XFORM( HOUSE, XFORM_MATRIX )
```

(continued on next page)

Example 9–2 (Cont.): The Effects of a Segment Transformation

```
C     Update the screen to show the transformed house.
❸     CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

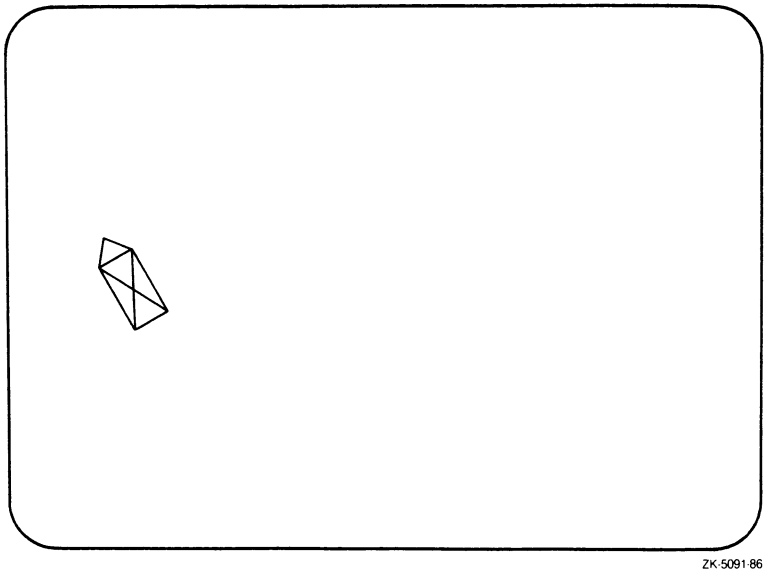
❶ This call selects a normalization transformation with the new viewport. DEC GKS transforms all output in segments to the new viewport. For the existence of the segment, the segment's primitives are clipped at this viewport boundary.

❷ Using a VT241, you cannot dynamically alter the segment currently displayed on the workstation surface by calling GKS$SET_SEG_XFORM. Calling this function at this point in the program changes the segment transformation in the segment state list, and sets flags in the workstation state list telling DEC GKS that the surface is out of date and that an update is necessary.

❸ Calling GKS$UPDATE_WS updates the position of the image on the workstation surface. The function GKS$REDRAW_SEG_ON_WS accomplishes the same task. Using both functions, all output *not* contained in segments is lost.

Figure 9–5 shows the screen of a VT241 terminal after the program has run to completion.

Figure 9–5: The Effects of a Segment Transformation—VT241



ZK-5091-86

In some applications, you may want to have more control over the order in which DEC GKS transforms segments. Simply, you may want to transform the segment in some other order besides scaling, then rotating, and finally translation. You can accomplish this task by calling GKS$ACCUM_XFORM_MATRIX.

For example, you may wish to translate, then scale, and finally rotate the segment. Assuming all of the variable declarations of the last code example, review the following code.

```
      .
      .
      .
C     To change the normal transformation order, set the translation...
      CALL GKS$EVAL_XFORM_MATRIX( NULL, NULL, VECTOR_X,
     * VECTOR_Y, NULL, 1.0, 1.0,
     * GKS$K_COORDINATES_WC, XFORM_MATRIX_1 )
      .
      .
      .
C     Accumulate the scaling...
      CALL GKS$ACCUM_XFORM_MATRIX( XFORM_MATRIX_1, FIXED_X, FIXED_Y,
     * NULL, NULL, NULL, SCALE_X, SCALE_Y, GKS$K_COORDINATES_WC,
     * XFORM_MATRIX_2 )
      .
      .
      .
C     And add the rotation...
      CALL GKS$ACCUM_XFORM_MATRIX( XFORM_MATRIX_2, FIXED_X, FIXED_Y,
     * NULL, NULL, ROTATION, 1.0, 1.0, GKS$K_COORDINATES_WC,
     * XFORM_MATRIX_3 )
      .
      .
      .
```

If you pass all three values to GKS$EVAL_XFORM_MATRIX, DEC GKS scales, then rotates, and finally translates. Using the two calls to GKS$ACCUM_XFORM_MATRIX, you force DEC GKS to translate, scale, and then rotate the segment, in that order.

9.4.4.1 Normalization and Segment Transformations, and Clipping

When you generate an output primitive during segment creation, DEC GKS stores the primitive, the currently associated output attributes, the current clipping rectangle (the current normalization viewport), and the pick identifier value (refer to Section 9.1.1).

When DEC GKS generates one of the primitives in a given segment, the primitive is transformed by the current normalization transformation, then the primitive is transformed by the specified segment transformation, and finally, if clipping was enabled before you generated the segment primitive (the default clipping status), the primitive is clipped at the *stored* normalization viewport boundary, not necessarily the *current* normalization viewport boundary.

If clipping is *not* enabled at the time you generate an output primitive during segment creation, DEC GKS stores the default normalization viewport ([0,1] x [0,1]) as the clipping rectangle for the generated primitive.

Consequently, when you translate a segment's position using GKS$EVAL_XFORM_MATRIX or GKS$ACCUM_XFORM_MATRIX, and if the segment crosses the viewport boundary, whether DEC GKS clips the primitives depends on the status of the clipping flag at the time of primitive generation.

Example 9–3 implements the normalization transformation, implements the segment transformation, and then clips the segment at the viewport boundary. Following the program example, Figure 9–6 illustrates the effects of the example on a VT241.

Example 9–3: Segment Transformations and Clipping

```
C     This program transforms the house contained in a segment and
C     clips it at the stored normalization viewport (clipping
C     rectangle).
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE, NUM_POINTS
      REAL VECTOR_Y, NULL, XFORM_MATRIX( 6 ), NO_CHANGE
      REAL PX ( 9 ), PY ( 9 )
      DATA VECTOR_Y / 0.1 /, NULL / 0.0 /
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE / 1 /, NO_CHANGE / 1.0 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Set the clipping boundary to be the lower left corner of the
C     default normalization viewport.
      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
(1)   CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
C     Even though clipping is the default, specify it to be clear.
(2)   CALL GKS$SET_CLIPPING( GKS$K_CLIP )

      CALL GKS$CREATE_SEG( HOUSE )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Set the current clipping rectangle to be the unity normalization
C     viewport ([0,1] X [0,1]). DEC GKS uses the stored viewport (lower
C     left corner), not this viewport, to clip the segment primitive.
      CALL GKS$SELECT_XFORM( 0 )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
```

(continued on next page)

Example 9–3 (Cont.): Segment Transformations and Clipping

```
C     Translate the house's position upwards by 1 world coordinate.
      CALL GKS$EVAL_XFORM_MATRIX( NULL, NULL, NULL,
     * VECTOR_Y, NULL, NO_CHANGE, NO_CHANGE,
     * GKS$K_COORDINATES_WC, XFORM_MATRIX )

C     Transform the segment and update the screen.
      CALL GKS$SET_SEG_XFORM( HOUSE, XFORM_MATRIX )
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

C     Pause.  Type RETURN when you are finished viewing the picture.
      READ(5,*)

C     Translate the house's position upwards by 1 world coordinate.
      CALL GKS$ACCUM_XFORM_MATRIX( XFORM_MATRIX, NULL, NULL,
     * NULL, VECTOR_Y, NULL, NO_CHANGE, NO_CHANGE,
     * GKS$K_COORDINATES_WC, XFORM_MATRIX )

C     Transform the segment and update the screen.
      CALL GKS$SET_SEG_XFORM( HOUSE, XFORM_MATRIX )
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

C     Pause.  Type RETURN when you are finished viewing the picture.
      READ(5,*)

C     Again, translate the house's position upwards by 1 world coordinate.
      CALL GKS$ACCUM_XFORM_MATRIX( XFORM_MATRIX, NULL, NULL,
     * NULL, VECTOR_Y, NULL, NO_CHANGE, NO_CHANGE,
     * GKS$K_COORDINATES_WC, XFORM_MATRIX )

C     Transform the segment.
      CALL GKS$SET_SEG_XFORM( HOUSE, XFORM_MATRIX )

C     Update the surface to initiate the change.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ This call selects a normalization transformation with the new viewport.

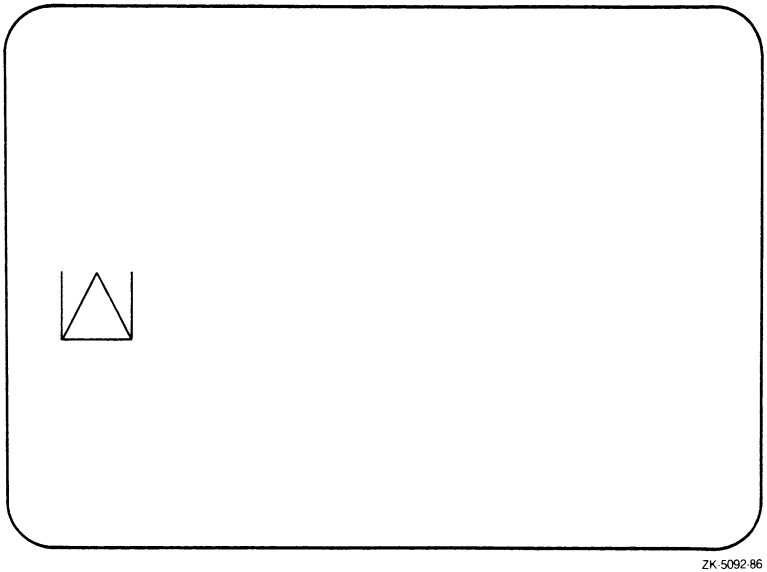
❷ Enabling clipping (the default setting) stores the new clipping window with any subsequently generated output primitives.

Although this program generates only one output primitive, you can change the clipping flag every time you output a primitive depending on the needs of your application. The current clipping rectangle is bound to subsequently generated primitives until you change the rectangle.

❸ This call to GKS$EVAL_XFORM_MATRIX alters the Y translation component so that the top portion of the house extends past the clipping rectangle.

On the VT241, you need to update the workstation surface to see changes in a segment transformation. Figure 9–6 shows the screen of a VT241 terminal after the program has run to completion.

Figure 9–6: Segment Transformations and Clipping—VT241



ZK-5092-86

During transformation, a segment's primitives may exceed the default normalization viewport, defined as ([0,1] x [0,1]). DEC GKS can store segments that exceed the default normalization viewport in NDC space.

However, even though DEC GKS can store segments that exceed the default normalization viewport boundary, those portions cannot be displayed on the surface of the workstation. DEC GKS clips all pictures at least at that workstation window boundary during the workstation transformation, and the maximum workstation window is ([0,1] x [0,1]) on the NDC plane.

9.4.4.2 Implementing Multiple Transformations

To understand the most complex combination of transformations, review the following code example:

```
                .
                .
                .
  C     These lines of code insert one segment's primitives into
  C     an open segment.
❶       CALL GKS$SELECT_XFORM( NORM )
❹       CALL GKS$SET_SEG_XFORM( SEG_ONE, MATRIX_ONE )
❺       CALL GKS$SET_CLIPPING( GKS$K_CLIP )
❷       CALL GKS$SET_SEG_XFORM( SEG_TWO, MATRIX_TWO )

        CALL GKS$CREATE_SEG( SEG_ONE )
❸       CALL GKS$INSERT_SEG( SEG_TWO, MATRIX_THREE )
        CALL CLOSE_SEG()
                .
                .
                .
```

In the previous example, there are four different transformations involved in the creation of the segment SEG_ONE. DEC GKS implements those transformations and clipping, in a cumulative fashion, in the following order:

❶ The first transformation that DEC GKS implements is always the current normalization transformation (NORM).

❷ If you are inserting a segment from WISS, DEC GKS always implements the inserted segment's transformation (MATRIX_TWO).

❸ Again, if you are inserting a segment from WISS, DEC GKS always implements the transformation specified in the call to GKS$INSERT_SEG, which is called the *insertion matrix* (MATRIX_THREE).

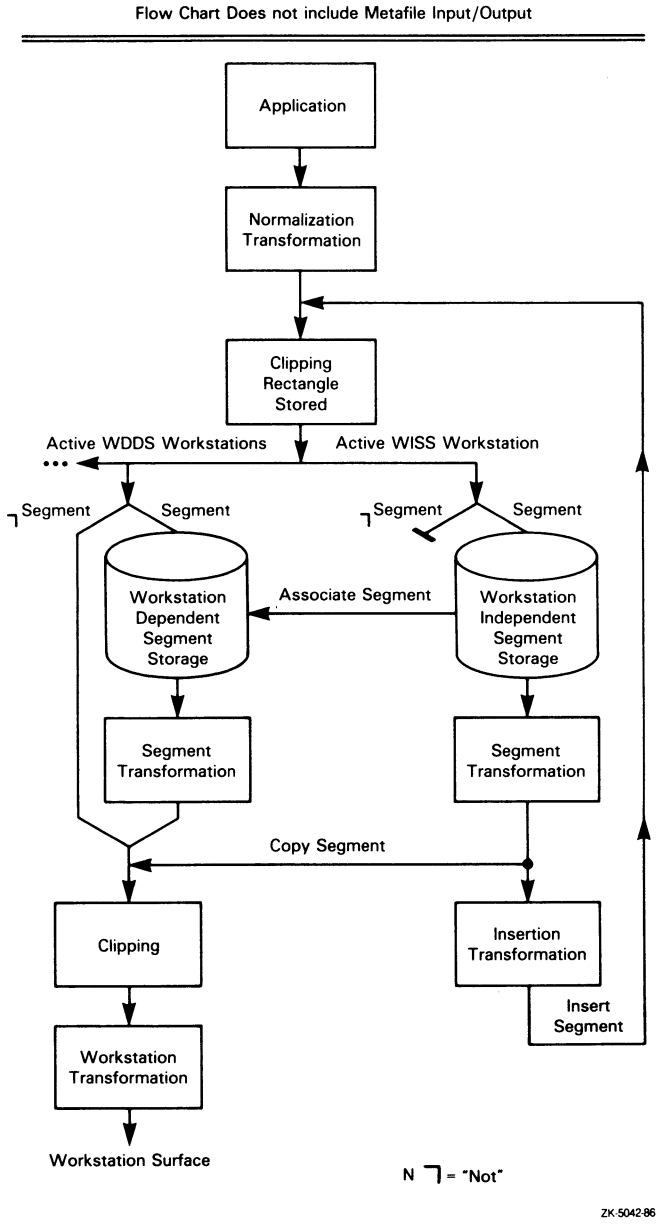
❹ Lastly, DEC GKS implements the transformation of the created segment (MATRIX_ONE).

❺ After all normalization and segment transformations are implemented, DEC GKS clips at the normalization viewport stored with the primitives in SEG_ONE (the viewport associated with NORM).

After the normalization and segment transformations, DEC GKS always applies a workstation transformation to the picture. DEC GKS clips all pictures at the workstation viewport boundary.

Figure 9–7 illustrates the transformation and clipping pipeline.

Figure 9–7: The Transformation and Clipping Pipeline



9.4.5 Visibility

This segment attribute determines if the segment is visible on the workstation surface. By default, DEC GKS segments are visible (GKS$K_VISIBLE).

Visibility can be used to hide a segment from a user until the segment is needed. For instance, segment visibility is a useful way to control the displaying of messages and menus, although GKS$MESSAGE and GKS$REQUEST_CHOICE can perform the same task.

By default, the visibility segment attribute is set to GKS$K_VISIBLE. Keep in mind that a segment must be both visible and detectable in order to pick that segment during pick input (refer to Chapter 8, Input Functions).

9.5 Segment Inquiries

The following list presents the inquiry functions that you can use to obtain segment information when writing device-independent code:

GKS$INQ_ACTIVE_WS
GKS$INQ_CLIP
GKS$INQ_DYN_MOD_SEG_ATTB
GKS$INQ_LEVEL
GKS$INQ_NAME_OPEN_SEG
GKS$INQ_OPEN_WS
GKS$INQ_OPERATING_STATE
GKS$INQ_PICK_ID
GKS$INQ_SEG_ATTB
GKS$INQ_SEG_NAMES
GKS$INQ_SEG_NAMES_ON_WS
GKS$INQ_SET_ASSOC_WS
GKS$INQ_WS_CATEGORY
GKS$INQ_WS_DEFER_AND_UPDATE
GKS$INQ_WS_MAX_NUM
GKS$INQ_WS_STATE
GKS$INQ_WS_TYPE

For more information concerning device-independent programming, refer to the *DEC GKS User Manual.*

9.6 Function Descriptions

This section describes the DEC GKS segment functions in detail.

ACCUMULATE TRANSFORMATION MATRIX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$ACCUM_XFORM_MATRIX accepts a specified transformation matrix, concatenates new segment transformation component values, and then writes the accumulated transformation to the last argument of the function. The only way to achieve a cumulative effect from successive segment transformations is to use GKS$ACCUM_XFORM_MATRIX.

See Section 9.4.4.1 for a detailed description of segment transformation and transformation matrixes.

Syntax

GKS$ACCUM_XFORM_MATRIX *(first_xform_matrix, fixed_point_x, fixed_point_y, translation_vec_x, translation_vec_y, rotation, scale_x, scale_y, type_of_coordinates, accumulated_xform_matrix)*

GACTM *(matrix, fx, fy, x_vector, y_vector, rotate, scale_x, scale_y, wc_ndc, nmatrix)*

gaccumtran *(segtran, ppoint, pshift, angle, pscale, coord, result)*

Arguments

first_xform_matrix

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

This argument is a six-element transformation matrix created previously by a call to either GKS$EVAL_XFORM_MATRIX or GKS$ACCUM_XFORM_MATRIX. You declare this argument as a six-element, single dimension array.

fixed_point_x
fixed_point_y

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments are the coordinates of the fixed point, used as the axis point during segment scaling and rotation. You can express this point as a world coordinate point or as an NDC point depending on the value you passed to the argument type_of_coordinates.

If you do not wish to scale or rotate the segment, you can pass the value 0.0 for both arguments.

translation_vec_x
translation_vec_y

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments express the coordinate values to be added to all segment X and Y values in order to alter the position of the segment. You can express these values as world coordinate values or as NDC values depending on the value you passed to the argument type_of_coordinates.

If you do not wish to translate the segment, you can pass the value 0.0 for both arguments.

rotation

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument is the measure of the angle of segment rotation around the fixed-point axis, in radians. To calculate radians, use the formula *360 degrees = 2*pi radians.*

If you do not wish to rotate the segment, you can pass the value 0.0 for this argument.

scale_x
scale_y

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments are the real number values that are multiplied times all of the segment's X and Y coordinates except the fixed point, to determine the new X and Y positions of the scaled segment.

If you do not wish to scale the segment, you can pass the value 1.0 for these arguments.

type_of_coordinates

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the flag that determines whether DEC GKS uses world coordinates or NDC coordinate values for the fixed point and translation vector arguments. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_COORDINATES_WC	The fixed point and translation vectors are world coordinate values.
1	GKS$K_COORDINATES_NDC	The fixed point and translation vectors are NDC values.

Use caution when specifying GKS$K_COORDINATES_WC. DEC GKS uses the *current* normalization transformation to transform the fixed point from world coordinates to NDC values. The current normalization transformation might not be the same as the one used during primitive generation. If the current normalization transformation is different, the result may be unexpected.

accumulated_xform_matrix

data type: **array (real)**
access: **write-only**
mechanism: **by reference**

This argument is the six-element transformation matrix that results from the concatenation of the new scaling, rotation, and translation component values with the argument first_xform_matrix. You can use this value as an argument to GKS$SET_SEG_XFORM to establish a cumulative segment transformation. You declare this argument as a six-element, single-dimension array.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****

Program Example

Example 9–4 illustrates the use of the function GKS$ACCUM_XFORM_MATRIX. Following the program example, Figure 9–8 illustrates the program's effect on a VT241 workstation.

Example 9–4: Showing the Cumulative Effect of GKS$ACCUM_XFORM_MATRIX

```
C     This program transforms the house contained in a segment.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE, NUM_POINTS
      REAL VECTOR_Y, NULL, XFORM_MATRIX( 6 ), NO_CHANGE
      REAL PX ( 9 ), PY ( 9 )
      DATA VECTOR_Y / 0.1 /, NULL / 0.0 /
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9, .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE / 1 /, NO_CHANGE / 1.0 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )


      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
(1)   CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$SET_CLIPPING( GKS$K_NOCLIP )

      CALL GKS$CREATE_SEG( HOUSE )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Shift the house upwards by 1 world coordinate.
      CALL GKS$EVAL_XFORM_MATRIX( NULL, NULL, NULL,
     * VECTOR_Y, NULL, NO_CHANGE, NO_CHANGE,
     * GKS$K_COORDINATES_WC, XFORM_MATRIX )

C     Transform the segment and update the screen.
(2)   CALL GKS$SET_SEG_XFORM( HOUSE, XFORM_MATRIX )
(3)   CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)
```

(continued on next page)

Example 9–4 (Cont.): Showing the Cumulative Effect of GKS$ACCUM_XFORM_MATRIX

```
C     Shift the house upwards by 1 more world coordinate.
❹     CALL GKS$ACCUM_XFORM_MATRIX( XFORM_MATRIX, NULL, NULL,
     * NULL, VECTOR_Y, NULL, NO_CHANGE, NO_CHANGE,
     * GKS$K_COORDINATES_WC, XFORM_MATRIX )

C     Transform the segment the segment and update the screen.
      CALL GKS$SET_SEG_XFORM( HOUSE, XFORM_MATRIX )
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Again, shift the house upwards by 1 more world coordinate.
      CALL GKS$ACCUM_XFORM_MATRIX( XFORM_MATRIX, NULL, NULL,
     * NULL, VECTOR_Y, NULL, NO_CHANGE, NO_CHANGE,
     * GKS$K_COORDINATES_WC, XFORM_MATRIX )

C     Transform the segment.
      CALL GKS$SET_SEG_XFORM( HOUSE, XFORM_MATRIX )

C     Update the surface to initiate the change.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ This call selects a normalization transformation with the new viewport. DEC GKS transforms all output in segments to the new viewport.

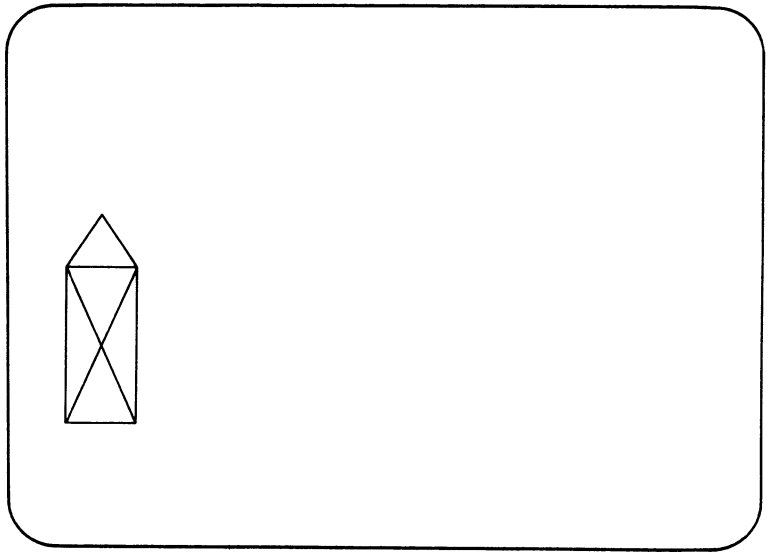
❷ Using the VT241, you cannot dynamically alter the segment, currently displayed on the workstation surface, by calling GKS$SET_SEG_XFORM. Calling this function at this point in the program changes the segment transformation in the segment state list, and sets flags in the workstation state list telling DEC GKS that the surface is out of date and that an update is necessary.

❸ Calling GKS$UPDATE_WS updates the position of the image on the workstation surface. The function GKS$REDRAW_SEG_ON_WS accomplishes the same task. Using both functions, all output *not* contained in segments is lost.

❹ Using GKS$ACCUM_XFORM_MATRIX, you can add transformation components to a previously set transformation. In this program, the house gradually moves upward, one Y world coordinate point at a time, using GKS$ACCUM_XFORM_MATRIX.

Figure 9–8 shows the screen of a VT241 terminal after the program has run to completion.

Figure 9–8: The Cumulative Effect of GKS$ACCUM_XFORM_ MATRIX—VT241



ASSOCIATE SEGMENT WITH WORKSTATION

Operating States: WSOP, WSAC

Description

The function GKS$ASSOC_SEG_WITH_WS takes a segment stored in workstation independent segment storage (WISS), and stores the segment on the specified workstation.

See Section 9.2 for more information concerning WISS.

Syntax

GKS$ASSOC_SEG_WITH_WS *(workstation_id, segment_name)*

GASGWK *(workstation_id, segment_name)*

gassocsegws *(workstation_id, segment_name)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value associated with an open or active workstation.

segment_name

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that identifies a segment that is stored on a WISS workstation.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
6	GKS$_ERROR_6	GKS not in proper state: GKS shall be either in the state WSOP or in the state WSAC in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
27	GKS$_ERROR_27	Workstation Independent Segment Storage is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
124	GKS$_ERROR_124	Specified segment does not exist on Workstation Independent Storage in routine ****

Program Example

Refer to Example 9–1 in this chapter for a program example using a call to GKS$ASSOC_SEG_WITH_WS.

CLOSE SEGMENT

Operating States: CLOSE SEGMENT

Description

The function GKS$CLOSE_SEG closes a segment. After you call this function, you can no longer add output primitives to that segment. You cannot reopen a segment.

Calling this function changes the DEC GKS operating state from GKS$K_SGOP (segment open) to GKS$K_WSAC (at least one workstation active).

Syntax

GKS$CLOSE_SEG *()*

GCLSG *()*

gcloseseg *()*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
4	GKS$_ERROR_4	GKS not in proper state: GKS shall be in the state SGOP in routine ****

Program Example

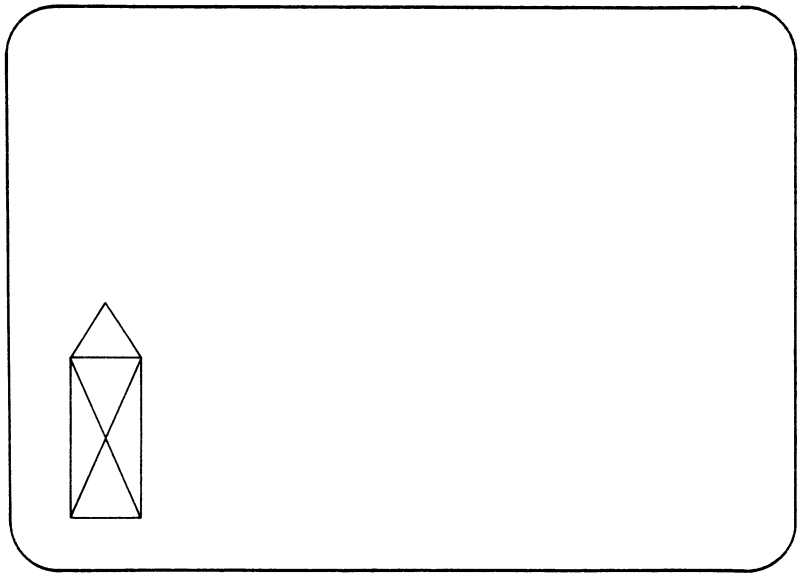
Example 9–5 illustrates the use of the function GKS$CLOSE_SEG. Following the program example, Figure 9–9 illustrates the effect of this example on a VT241.

Example 9–5: Drawing a House and Placing It in a Segment

```
C     This program draws a house in the lower left corner of the
C     screen.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE, NUM_POINTS
      REAL PX ( 9 ), PY ( 9 )
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE / 1 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )

      CALL GKS$CREATE_SEG( HOUSE )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

Figure 9–9 shows the screen of a VT241 terminal after the program has run to completion.

Figure 9–9: House in the Lower Left Corner of the Screen—VT241



ZK-5098-86

COPY SEGMENT TO WORKSTATION

Operating States: WSOP, WSAC

Description

The function GKS$COPY_SEG_TO_WS takes a segment stored in workstation independent segment storage (WISS), transforms it according to its segment transformation, clips it according to the stored clipping rectangles for each primitive, and then copies its primitives to the specified workstation. The specified workstation does *not* store the primitives in a segment.

See Section 9.2 for more information concerning WISS.

Syntax

GKS$COPY_SEG_TO_WS *(workstation_id, segment_name)*

GCSGWK *(workstation_id, segment_name)*

gcopysegws *(workstation_id, segment_name)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value associated with an open or active workstation. This value cannot be GKS$K_WSTYPE_WISS.

segment_name

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the integer value that identifies a segment. This segment must be stored on a WISS workstation.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-94	DECGKS$_ERROR_NEG_94	Internal GKS error: Corrupted segment memory in routine ****
6	GKS$_ERROR_6	GKS not in proper state: GKS shall be either in the state WSOP or in the state WSAC in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
27	GKS$_ERROR_27	Workstation Independent Segment Storage is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****

Error Number	Completion Status Code	Message
36	GKS$_ERROR_36	Specified workstation is of category Workstation Independent Segment Storage in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
124	GKS$_ERROR_124	Specified segment does not exist on Workstation Independent Storage in routine ****

Program Example

Refer to Example 9–1 in this chapter for a program example using a call to GKS$COPY_SEG_TO_WS.

CREATE SEGMENT

Operating States: WSCA

Description

The function GKS$CREATE_SEG opens a segment on all active workstations and names that segment. All subsequent calls to output functions add output primitives to the segment until you call GKS$CLOSE_SEG. DEC GKS stores output primitives, output attributes, and the current clipping rectangle for each of the primitives. You cannot nest a call to GKS$CREATE_SEG within the creation of another segment. See Section 9.1 for more information.

DEC GKS must be in the GKS$K_WSAC (at least one workstation active) operating state to call this function. After calling GKS$CREATE_SEG, the operating state changes from GKS$K_WSAC to GKS$K_SGOP (segment open).

Syntax

GKS$CREATE_SEG *(segment_name)*

GCRSG *(segment_name)*

gcreateseg *(segment_name)*

Arguments

segment_name

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that identifies a stored segment. You cannot use the same identifier to name two different segments. Using DEC GKS, you must use positive integers as segment names.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
3	GKS$_ERROR_3	GKS not in proper state: GKS shall be in the state WSAC in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
121	GKS$_ERROR_121	Specified segment name is already in use in routine ****

Program Example

Refer to Example 9–5 in this chapter for a program example using a call to GKS$CREATE_SEG.

DELETE SEGMENT

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$DELETE_SEG deletes the specified segment from all workstations storing that segment. You can delete any defined segment, but you cannot delete an open segment.

Calling this function deletes the specified segment's state list.

Syntax

GKS$DELETE_SEG *(segment_name)*

GDSG *(segment_name)*

gdelseg *(segment_name)*

Arguments

segment_name

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that identifies a stored segment.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
122	GKS$_ERROR_122	Specified segment does not exist in routine ****
125	GKS$_ERROR_125	Specified segment is open in routine ****

Program Example

Example 9–6 illustrates the use of the function GKS$DELETE_SEG.

Example 9–6: Deleting Segments on All Open and Active Workstations

```
C     This program draws a house in the lower left corner of the
C     screen and then deletes it.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE, WISS,
     * NUM_POINTS
      REAL PX ( 9 ), PY ( 9 )
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE / 1 /, WISS / 2 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
(1)   CALL GKS$OPEN_WS( WISS, GKS$K_CONID_DEFAULT,
     * GKS$K_WSTYPE_WISS )

      CALL GKS$ACTIVATE_WS( WISS )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )

C     DEC GKS stores this segment on WS_ID and the WISS workstation.
      CALL GKS$CREATE_SEG( HOUSE )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )

C     Close the currently open segment.
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Delete the segment from all open and active workstations.
(2)   CALL GKS$DELETE_SEG( HOUSE )
```

(continued on next page)

Example 9–6 (Cont.): Deleting Segments on All Open and Active Workstations

```
C     You have to update the screen to delete the output primitives
C     from the surface.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$DEACTIVATE_WS( WISS )
      CALL GKS$CLOSE_WS( WISS )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ This line of code opens a WISS workstation. From this point forward, anytime DEC GKS stores output in a segment on the VT241, it stores the same output in the same segment on the WISS workstation.

❷ After this line of code is executed, you cannot copy the segment HOUSE from the WISS workstation to the workstation WS_ID, since this line deletes the segment from all workstations.

DELETE SEGMENT FROM WORKSTATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$DELETE_SEG_FROM_WS deletes the segment from the specified workstation storing that segment. You can delete any defined segment, but you cannot delete an open segment.

If you delete the segment from the last workstation supporting a given segment, calling this function deletes the specified segment's state list, which has the same effect as a call to the function GKS$DELETE_SEG.

Syntax

GKS$DELETE_SEG_FROM_WS *(workstation_id, segment_name)*

GDSGWK *(workstation_id, segment_name)*

gdelseg *(workstation_id, segment_name)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value associated with an open or active workstation.

segment_name

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the integer value that identifies a stored segment.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC, or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
33	GKS$_ERROR_33	Specified workstation is of category MI in routine ****
35	GKS$_ERROR_35	Specified workstation is of category INPUT in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
123	GKS$_ERROR_123	Specified segment does not exist on specified workstation in routine ****
125	GKS$_ERROR_125	Specified segment is open in routine ****

Program Example

Example 9–7 illustrates the use of the function GKS$DELETE_SEG_FROM_WS.

Example 9–7: Deleting Segments on a Specific Workstation

```
C     This program draws a house in the lower left corner of the
C     screen and then deletes it from the VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE, WISS,
     * NUM_POINTS
      REAL PX ( 9 ), PY ( 9 )
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE / 1 /, WISS / 2 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
(1)   CALL GKS$OPEN_WS( WISS, GKS$K_CONID_DEFAULT,
     * GKS$K_WSTYPE_WISS )

      CALL GKS$ACTIVATE_WS( WISS )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )

C     DEC GKS stores this segment on WS_ID and the WISS workstation.
      CALL GKS$CREATE_SEG( HOUSE )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Delete the segment from WS_ID and update the screen.
(2)   CALL GKS$DELETE_SEG_FROM_WS( WS_ID, HOUSE )
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )
```

(continued on next page)

Example 9–7 (Cont.): Deleting Segments on a Specific Workstation

```
C     Pause.  Type RETURN when you are finished
C     viewing the picture.
      READ(5,*)

C     Associate HOUSE in WISS with WS_ID.
❸     CALL GKS$ASSOC_SEG_WITH_WS( WS_ID, HOUSE )

C     Update the surface to initiate the change.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$DEACTIVATE_WS( WISS )
      CALL GKS$CLOSE_WS( WISS )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ This line of code opens a WISS workstation. From this point forward, anytime DEC GKS stores output in a segment on the VT241, it stores the same output in the same segment on the WISS workstation.

❷ After this line of code is executed, the segment HOUSE is deleted from the workstation WS_ID. However, the WISS workstation still retains the segment HOUSE.

❸ If you want the workstation WS_ID to store the segment HOUSE again, you have to associate the segment with the workstation. Afterwards, WS_ID stores the segment HOUSE.

EVALUATE TRANSFORMATION MATRIX

Operating States: GKOP, WSOP, WSAC, SGOP

Description

The function GKS$EVAL_XFORM_MATRIX accepts scaling, rotation, and translation component values, and then writes a transformation matrix to the last argument of the function. You specify the transformation matrix as an argument to GKS$SET_SEG_XFORM to establish a segment transformation.

See Section 9.4.4 for a detailed description of segment transformation and transformation matrixes.

Syntax

GKS$EVAL_XFORM_MATRIX *(fixed_point_x, fixed_point_y, translation_vec_x, translation_vec_y, rotation, scale_x, scale_y, type_of_coordinates, transformation_matrix)*

GEVTMK *(fx, fy, x_vector, y_vector, rotate, scale_x, scale_y, we_ndc, matrix)*

gevaltran *(ppoint, pshift, angle, pscale, coord, result)*

Arguments

fixed_point_x
fixed_point_y

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

These arguments are the coordinates of the fixed point, used as the only constant coordinate point during scaling and as the axis point during segment rotation. You can express this point as a world coordinate point or as an NDC point depending on the value you passed to the argument type_of_coordinates.

If you do not wish to scale or rotate the segment, you can pass the value 0 for both arguments.

translation_vec_x
translation_vec_y

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

These arguments express the coordinate values to be added to all segment X and Y values in order to alter the position of the segment. You can express these values as world coordinate values or as NDC values depending on the value you passed to the argument type_of_coordinates.

If you do not wish to translate the segment, you can pass the value 0 for both arguments.

rotation

data type:	**real**
access:	**read-only**
mechanism:	**by reference**

This argument is the measure of the angle of segment rotation around the fixed point axis, in radians. To calculate radians, use the formula *360 degrees = 2*pi.*

If you do not wish to rotate the segment, you can pass the value 0 for this argument.

scale_x
scale_y

data type: **real**
access: **read-only**
mechanism: **by reference**

These arguments are the real number values that are multiplied times all of the segment's X and Y coordinates except the fixed point, to determine the new X and Y positions of the scaled segment.

If you do not wish to scale the segment, you can pass the value 1.0 for these arguments.

type_of_coordinates

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the flag that determines whether GKS uses world coordinates or NDC coordinate values for the fixed point and shift vector arguments. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_COORDINATES_WC	The fixed point and shift vectors are world coordinate values.
1	GKS$K_COORDINATES_NDC	The fixed point and shift vectors are NDC values.

Use caution when specifying GKS$K_COORDINATES_WC. DEC GKS uses the *current* normalization transformation to transform the fixed point from world coordinates to NDC values. The current normalization transformation might not be the same as the one used during primitive generation. If the current normalization transformation is different, the result may be unexpected.

transformation_matrix

data type: **array (real)**
access: **write-only**
mechanism: **by reference**

This argument is the six-element transformation matrix that results from the evaluation of the scaling, rotation, and shifting component values. You can use this value as an argument to GKS$SET_SEG_XFORM to establish a segment transformation. You declare this argument as a six-element, single-dimension array.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC, or SGOP in routine ****

Program Example

Refer to Example 9–2 in this chapter for a program example using a call to GKS$EVAL_XFORM_MATRIX.

INSERT SEGMENT

Operating States: WSAC, SGOP

Description

The function GKS$INSERT_SEG takes a segment stored in a workstation Independent segment storage (WISS) workstation, transforms it according to the segment's current transformation, and transforms it again according to the transformation matrix specified in the call to GKS$INSERT_SEG (the effect of the two transformations are cumulative). Then, if the operating state is GKS$K_WSAC (at least one workstation active), GKS$INSERT_SEG copies the segment's output primitives onto the surface of all active workstations.

If you call GKS$INSERT_SEG when another segment is open (DEC GKS operating state GKS$K_SGOP), DEC GKS inserts the specified segment's output primitives into the open segment.

Whether or not you have a segment open at the time of the call to GKS$INSERT_SEG, the active workstations do *not* store the inserted segment primitives as a segment; the workstations only generate the inserted segment's primitives.

DEC GKS applies only the current clipping rectangle to the inserted primitives, and if a segment is currently open, applies the segment attributes of the open segment (transformation, highlighting, visibility, and so forth) to the inserted primitives. The inserted primitive's attributes (line type, text expansion factor, and so forth) remain unchanged.

Syntax

GKS$INSERT_SEG *(segment_name, insertion_xform_matrix)*

GINSG *(segment_name, matrix)*

ginsertseg *(segment_name, segtran)*

Arguments

segment_name

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the integer value that identifies a stored segment.

insertion_xform_matrix

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

This argument is the six-element insertion transformation matrix. If a transformation has been specified for the segment to be inserted, DEC GKS calculates the accumulated effect of the segment transformation and then the insertion transformation, in that order. You can formulate an insertion transformation using either GKS$EVAL_XFORM_MATRIX or GKS$ACCUM_XFORM_MATRIX.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
5	GKS$_ERROR_5	GKS not in proper state: GKS shall be either in the state WSAC or in the state SGOP in routine ****
27	GKS$_ERROR_27	Workstation Independent Segment Storage is not open in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****

Error Number	Completion Status Code	Message
124	GKS$_ERROR_124	Specified segment does not exist on Workstation Independent Segment Storage in routine ****
125	GKS$_ERROR_125	Specified segment is open in routine ****

Program Example

Example 9–8 illustrates the use of the function GKS$INSERT_SEG. Following the program example, Figure 9–10 illustrates the program's effect on a VT241 workstation.

Example 9–8: Inserting a Segment's Primitives into Another Segment

```
C     This program draws a house in the lower left corner of the
C     screen and then inserts that house into other segments.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE_1, HOUSE_2,
     * WISS, NUM_POINTS, UPPER_LEFT_CORNER,
     * UPPER_RIGHT_CORNER, LOWER_RIGHT_CORNER
      REAL PX ( 9 ), PY ( 9 ), NULL, NO_CHANGE,
     * XFORM_MATRIX( 6 ), UP, RIGHT
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE_1 / 1 /, HOUSE_2 / 2 /, WISS / 2 /,
     * NULL / 0.0 /, NO_CHANGE / 1.0 /, WS_ID / 1 /,
     * UP / 0.5 /, RIGHT / 0.5 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$OPEN_WS( WISS, GKS$K_CONID_DEFAULT,
     * GKS$K_WSTYPE_WISS )

      CALL GKS$ACTIVATE_WS( WS_ID )
      CALL GKS$ACTIVATE_WS( WISS )

(1)   CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
```

(continued on next page)

Example 9–8 (Cont.): Inserting a Segment's Primitives into Another Segment

```
C     Create a segment in the lower left corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_1 )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Deactivate WISS so that no other segments are stored there.
      CALL GKS$DEACTIVATE_WS( WISS )

C     Turn off the clipping so that transformed houses are visible.
      CALL GKS$SET_CLIPPING( GKS$K_NOCLIP )

C     Change the matrix value.
      CALL GKS$EVAL_XFORM_MATRIX( NULL, NULL, RIGHT, NULL,
     * NULL, NO_CHANGE, NO_CHANGE, GKS$K_COORDINATES_NDC,
     * XFORM_MATRIX )
```

(continued on next page)

Example 9–8 (Cont.): Inserting a Segment's Primitives into Another Segment

```
C     Create a segment in the lower right corner by inserting
C     HOUSE_1's primitives into HOUSE_2.
      CALL GKS$CREATE_SEG( HOUSE_2 )
❷     CALL GKS$INSERT_SEG( HOUSE_1, XFORM_MATRIX )
      CALL GKS$CLOSE_SEG()

C     Change the matrix value.
      CALL GKS$EVAL_XFORM_MATRIX( NULL, NULL, NULL, UP,
     * NULL, NO_CHANGE, NO_CHANGE, GKS$K_COORDINATES_NDC,
     * XFORM_MATRIX )

C     Just insert the primitives in the upper left corner using
C     GKS$INSERT_SEG.
❸     CALL GKS$INSERT_SEG( HOUSE_1, XFORM_MATRIX )

C     Change the matrix value.
      CALL GKS$EVAL_XFORM_MATRIX( NULL, NULL, RIGHT, UP,
     * NULL, NO_CHANGE, NO_CHANGE, GKS$K_COORDINATES_NDC,
     * XFORM_MATRIX )

C     Just insert the primitives in the upper right corner using
C     GKS$INSERT_SEG.
      CALL GKS$INSERT_SEG( HOUSE_1, XFORM_MATRIX )

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Redraw all of the segments.  House primitives at the top are deleted.
❹     CALL GKS$REDRAW_SEG_ON_WS( WS_ID )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WISS )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ These lines of code establish the normalization window to be the lower left corner of the NDC space.

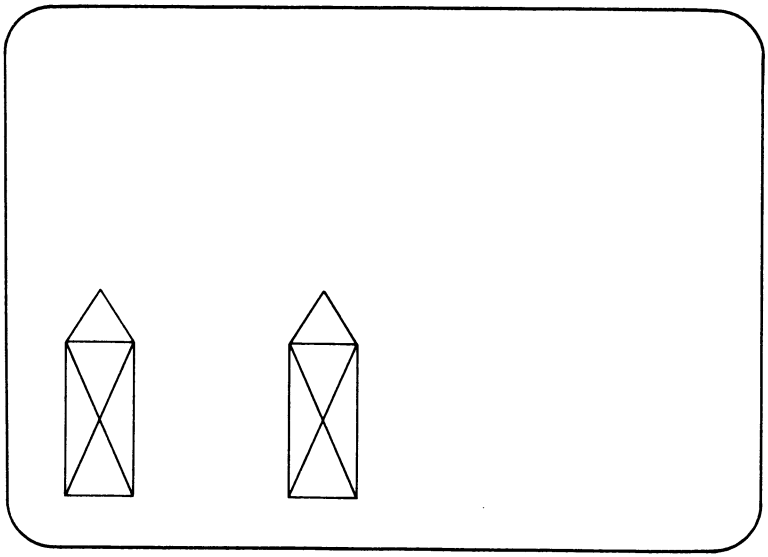
❷ Using GKS$EVAL_XFORM_MATRIX, you can create the transformation matrix that you need to pass to GKS$INSERT_SEG as an argument. This matrix specifies a position translation of 0.5 NDC points to the right. When this matrix is passed to GKS$INSERT_SEG while a segment is open, the house's primitives are transformed and made a part of the open segment.

❸ Inserting segments when the DEC GKS operating state is GKS$K_WSAC causes the output primitives to be written to the workstation surface, but the primitives are not stored in a segment. These segment primitives are translated 0.5 NDC points in an upwards direction.

❹ The call to GKS$REDRAW_SEG_ON_WS redraws all segments and deletes all primitives outside of segments.

Figure 9–10 shows the screen of a VT241 terminal after the program has run to completion.

Figure 9–10: Inserting a Segment's Primitives into Another Segment—VT241



RENAME SEGMENT

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$RENAME_SEG changes the segment name from its former name to a new name. Once you have renamed a segment, you can reuse the old segment name.

Syntax

GKS$RENAME_SEG *(old_name, new_name)*

GRENSG *(old_name, new_name)*

grenameseg *(old, new)*

Arguments

old_name

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that was used to identify the segment. After the call to GKS$RENAME_SEG, you are free to use this identifier to name another segment.

new_name

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that now identifies the segment.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
121	GKS$_ERROR_121	Specified segment name is already in use in routine ****
122	GKS$_ERROR_122	Specified segment does not exist in routine ****

Program Example

Example 9–9 illustrates the use of the function GKS$RENAME_SEG.

Example 9–9: Renaming a Segment

```
C     This program draws a house in the lower left corner of the
C     screen and then renames it.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE_1, HOUSE_2,
     * LOWER_RIGHT_CORNER, UPPER_LEFT_CORNER, HOUSE_3,
     * NUM_POINTS
      REAL PX ( 9 ), PY ( 9 )
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE_1 / 1 /, HOUSE_2 / 2 /, LOWER_RIGHT_CORNER / 2 /,
     * UPPER_LEFT_CORNER / 3 /, HOUSE_3 / 3 /, WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

❶     CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
      CALL GKS$SET_VIEWPORT( LOWER_RIGHT_CORNER, 0.5, 1.0, 0.0, 0.5 )
      CALL GKS$SET_VIEWPORT( UPPER_LEFT_CORNER, 0.0, 0.5, 0.5, 1.0 )

C     Create a segment in the lower left corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_1 )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Create a second segment in the lower right corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_2 )
      CALL GKS$SELECT_XFORM( LOWER_RIGHT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Create a third segment in the upper left corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_3 )
      CALL GKS$SELECT_XFORM( UPPER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Delete houses 1 and 2.
      CALL GKS$DELETE_SEG( HOUSE_1 )
      CALL GKS$DELETE_SEG( HOUSE_2 )
```

(continued on next page)

Example 9–9 (Cont.): Renaming a Segment

```
C     Redraw all of the segments.
      CALL GKS$REDRAW_SEG_ON_WS( WS_ID )

C     Since it is the only one left, rename HOUSE_3 to be HOUSE_1.
❷     CALL GKS$RENAME_SEG( HOUSE_3, HOUSE_1 )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ These lines of code establish three different normalization transformations in three corners of the NDC plane.

❷ After deleting segments, you may want to keep all closed segments in numerical order. You can use GKS$RENAME_SEG to maintain an orderly progression of segment names.

SET PICK ID

Operating States: GKOP, WSOP, WSAC, SGOP

Description

This function sets the *current pick identifier* value in the DEC GKS state list to be the number you specify. All subsequent output primitives stored in segments are assigned this value until you change it.

Setting pick identifiers allows you another level of naming sections within segments so that a user can pick portions of a segment without having to pick the whole segment.

NOTE

DEC GKS continues to recognize the last pick identifier specified, even after you close a segment. If you open another segment, DEC GKS continues to associate the current segment identifier with the newly output images. Consequently, if you specify a pick identifier in one segment, make sure that you set the pick identifier properly when opening another segment.

Syntax

SET PICK IDENTIFIER *(pick_id)*

GSPKID *(pick_id)*

gsetpickid *(pick_id)*

Arguments

pick_id

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the new pick identifier.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
8	GKS$_ERROR_8	GKS not in proper state; GKS shall be in one of the states GKOP, WSOP, WSAC, or SGOP
97	GKS$_ERROR_97	Pick identifier is invalid

Program Example

Example 9–10 illustrates the use of the function GKS$SET_PICK_ID. Following the program example, Figure 9–11 illustrates the program's effect on a VT241 workstation.

Example 9–10: Setting Pick Identifiers

```
C     This program initializes and requests pick input from a VT241.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, INITIAL_STATUS, SEGMENT,
     * PICK_ID, PROMPT_ECHO_TYPE, ERROR_STATUS,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE, INPUT_STATUS DEVICE_NUM, INPUT_CHOICE,
     * BOX_1, BOX_2, TRIANGLE_1, TRIANGLE_2, NUM_POINTS
❶     REAL ECHO_AREA(4), DATA_RECORD( 1 )
      REAL X_VALUES( 4 ), Y_VALUES( 4 )
      DATA WS_ID / 1 /, DEVICE_NUM / 1 /, BOX_1 / 1 /,
     * BOX_2 / 2 /, TRIANGLE_1 / 1 /, TRIANGLE_2 / 2 /,
     * NUM_POINTS / 4 /
      DATA X_VALUES / 0.1, 0.4, 0.1, 0.1 /
      DATA Y_VALUES / 0.3, 0.6, 0.6, 0.3 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )

❷     CALL GKS$CREATE_SEG( BOX_1 )
      CALL GKS$SET_PICK_ID( TRIANGLE_1 )
      CALL GKS$SET_FILL_COLOR_INDEX( 2 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      X_VALUES( 3 ) = 0.4
      Y_VALUES( 3 ) = 0.3
      CALL GKS$SET_PICK_ID( TRIANGLE_2 )
      CALL GKS$SET_FILL_COLOR_INDEX( 3 )
      CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
      CALL GKS$CLOSE_SEG()

❸     X_VALUES( 1 ) = 0.6
      X_VALUES( 2 ) = 0.9
      X_VALUES( 3 ) = 0.6
      X_VALUES( 4 ) = 0.6
      Y_VALUES( 3 ) = 0.6

      CALL GKS$SET_PICK_ID( TRIANGLE_1 )
```

(continued on next page)

Example 9–10 (Cont.): Setting Pick Identifiers

```
❹       CALL GKS$CREATE_SEG( BOX_2 )
        CALL GKS$SET_FILL_COLOR_INDEX( 2 )
        CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
        X_VALUES( 3 ) = 0.9
        Y_VALUES( 3 ) = 0.3
        CALL GKS$SET_PICK_ID( TRIANGLE_2 )
        CALL GKS$SET_FILL_COLOR_INDEX( 3 )
        CALL GKS$FILL_AREA( NUM_POINTS, X_VALUES, Y_VALUES )
        CALL GKS$CLOSE_SEG()

        CALL GKS$SET_SEG_DETECTABILITY( BOX_1, GKS$K_DETECTABLE )
        CALL GKS$SET_SEG_DETECTABILITY( BOX_2, GKS$K_DETECTABLE )

❺       CALL GKS$SET_TEXT_HEIGHT( 0.03 )
        CALL GKS$TEXT( 0.2, 0.45, '1' )
        CALL GKS$TEXT( 0.3, 0.45, '2' )
        CALL GKS$TEXT( 0.7, 0.45, '1' )
        CALL GKS$TEXT( 0.8, 0.45, '2' )

C       Declare a data length of one long word which will hold the
C       size of the pick prompt.
❻       RECORD_BUFFER_LENGTH = 4
        CALL GKS$INQ_PICK_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE,
     * ECHO_FLAG, INITIAL_STATUS, SEGMENT, PICK_ID, PROMPT_ECHO_TYPE,
     * ECHO_AREA, DATA_RECORD, RECORD_BUFFER_LENGTH,
     * RECORD_SIZE )

❼       SEGMENT = BOX_1
        PICK_ID = TRIANGLE_1
        PROMPT_ECHO_TYPE = 1
        INITIAL_STATUS = GKS$K_STATUS_NOPICK

❽       CALL GKS$INIT_PICK( WS_ID, DEVICE_NUM, INITIAL_STATUS,
     * SEGMENT, PICK_ID, PROMPT_ECHO_TYPE, ECHO_AREA,
     * DATA_RECORD, RECORD_BUFFER_LENGTH )

❾       CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )
```

(continued on next page)

Example 9–10 (Cont.): Setting Pick Identifiers

```
⓾      CALL GKS$REQUEST_PICK( WS_ID, DEVICE_NUM, INPUT_STATUS,
      * SEGMENT, PICK_ID )

C     Output the input choice number.
      WRITE(6,*) INPUT_STATUS, SEGMENT, PICK_ID

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ The data record is one real value that represents the size of the pick prompt (or, *aperture*) in device coordinates.

The echo area variable is an array of real numbers representing the rectangular echo area, in device coordinates. The echo area defines a portion of the workstation surface on which the prompt moves.

❷ This code creates a box on the left side of the workstation surface and places it in a segment. The code divides the box diagonally and sets pick identifiers for each of the created fill area triangles.

❸ This code resets the X and Y world coordinate values so that the coordinate values specify a new position for the next box.

❹ This code creates a box on the right side of the workstation surface and places it in a segment. The code divides the box diagonally and sets pick identifiers for each of the created triangles.

❺ This code labels the triangles by their pick identifiers.

❻ This code initializes the size of the data record. This variable is a modifiable variable passed to GKS$INQ_PICK_STATE, and you must initialize it before calling the inquiry function.

The function GKS$INQ_PICK_STATE initializes the variables needed by the input functions. The argument GKS$K_VALUE_REALIZED tells the graphics handler to pass the input values as they are implemented by the graphics handler, as opposed to the way that the application may have set the values (GKS$K_VALUE_SET).

The second to last argument specifies the length of the argument that is to contain the data record. After DEC GKS returns the data record, it modifies this argument to contain the length of the returned data record. By comparing the last two arguments, you can tell whether your data record variable was large enough to hold the entire data record.

❼ This code assigns new values to the input variables. For instance, the initial segment identifier has the value 1.

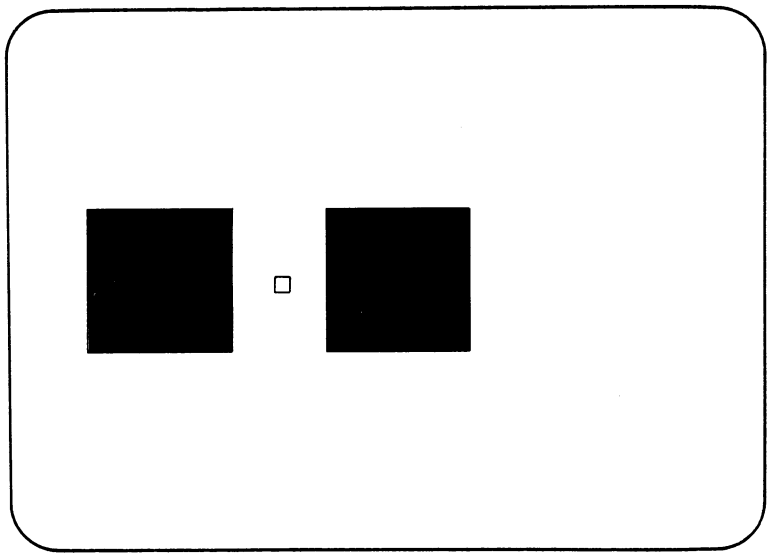
❽ The function GKS$INIT_PICK initializes the request for choice input.

❾ You must pass GKS$K_INPUT_MODE_REQUEST as an argument to the function GKS$SET_PICK_MODE, since the DEC GKS software does not support sample or event mode. You can use this function to enable (as in this example) or to disable input prompt echoing.

❿ The function GKS$REQUEST_PICK prompts the user for input. The segment and pick identifiers are written to the last arguments.

Figure 9–11 shows the screen of a VT241 terminal at the request for input.

Figure 9–11: Setting Pick Identifiers—VT241



SET SEGMENT DETECTABILITY

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_SEG_DETECTABILITY controls the segment attribute that determines whether or not the user can choose a segment during pick input. (A segment has to be both detectable and visible in order to be picked.)

Syntax

SET DETECTABILITY *(segment_name, detectability_flag)*

GSDTEC *(segment_name, detect)*

gsetdet *(segment_name, detectability)*

Arguments

segment_name

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that identifies a stored segment.

detectability_flag

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the flag that determines whether or not the user can pick the specified segment. By default, DEC GKS segments are GKS$K_UNDETECTABLE. This argument can be either of the following values or constants.

Value	Constant	Description
0	GKS$K_UNDETECTABLE	You cannot pick the segment.
1	GKS$K_DETECTABLE	You can pick the segment, if visible.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
122	GKS$_ERROR_122	Specified segment does not exist in routine ****

Program Example

Example 9–11 illustrates the use of the function GKS$SET_SEG_DETECTABILITY.

Example 9–11: Controlling the Detectability of Segments

```
C     This program draws a house in the lower left corner of the
C     screen and an undetectable house in the upper right corner.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE_1, HOUSE_2,
     * UPPER_RIGHT_CORNER, INITIAL_STATUS,
     * PICK_ID, PROMPT_ECHO_TYPE, ERROR_STATUS, SEGMENT_NAME,
     * INPUT_MODE, ECHO_FLAG, RECORD_BUFFER_LENGTH, RECORD_SIZE,
     * INPUT_STATUS, NUM_POINTS, DEVICE_NUM
      REAL PX ( 9 ), PY ( 9 ), ECHO_AREA( 4 ), DATA_RECORD( 1 )
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE_1 / 1 /, HOUSE_2 / 2 /, UPPER_RIGHT_CORNER / 2 /,
     * WS_ID / 1 /, DEVICE_NUM / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
      CALL GKS$SET_VIEWPORT( UPPER_RIGHT_CORNER, 0.5, 1.0, 0.5, 1.0 )

C     Create a segment in the lower left corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_1 )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Create a second segment in the upper right corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_2 )
      CALL GKS$SELECT_XFORM( UPPER_RIGHT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Make HOUSE_1 detectable.
(1)   CALL GKS$SET_SEG_DETECTABILITY( HOUSE_1, GKS$K_DETECTABLE )
```

(continued on next page)

Example 9–11 (Cont.): Controlling the Detectability of Segments

```
C     Inquire, initialize, set, and request pick input.
      RECORD_BUFFER_LENGTH = 4
❷     CALL GKS$INQ_PICK_STATE( WS_ID, DEVICE_NUM,
     * GKS$K_VALUE_REALIZED, ERROR_STATUS, INPUT_MODE,
     * ECHO_FLAG, INITIAL_STATUS, SEGMENT_NAME, PICK_ID,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH, RECORD_SIZE )

❸     SEGMENT_NAME = HOUSE_1
      PROMPT_ECHO_TYPE = 1
      DEVICE_NUM = 1
      ECHO_AREA( 1 ) = 0.0
      ECHO_AREA( 2 ) = 479.0
      ECHO_AREA( 3 ) = 0.0
      ECHO_AREA( 4 ) = 479.0

❹     CALL GKS$INIT_PICK( WS_ID, DEVICE_NUM,
     * GKS$K_STATUS_OK, SEGMENT_NAME, PICK_ID,
     * PROMPT_ECHO_TYPE, ECHO_AREA, DATA_RECORD,
     * RECORD_BUFFER_LENGTH )

❺     CALL GKS$SET_PICK_MODE( WS_ID, DEVICE_NUM,
     * GKS$K_INPUT_MODE_REQUEST, GKS$K_ECHO )

❻     CALL GKS$REQUEST_PICK( WS_ID, DEVICE_NUM, INPUT_STATUS,
     * SEGMENT_NAME, PICK_ID )

C     Output the input segment name.
❼     WRITE(6,*) INPUT_STATUS, SEGMENT_NAME

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ By setting the segment attribute of HOUSE_1 to GKS$K_DETECTABLE, you allow the segment to be picked during input.

❷ The function GKS$INQ_PICK_STATE obtains default values needed for pick input. For more information, refer to Chapter 12, Inquiry Functions.

❸ Setting the segment name to HOUSE_1 ensures an initial pick value of 1. Since HOUSE_2 is undetectable, you cannot set the initial segment name to be 2.

❹ The call to GKS$INIT_PICK initializes input values. For more information concerning this function, refer to Chapter 8, Input Functions.

❺ The call to GKS$SET_PICK_MODE controls the prompt echo. For more information concerning this function, refer to Chapter 8, Input Functions.

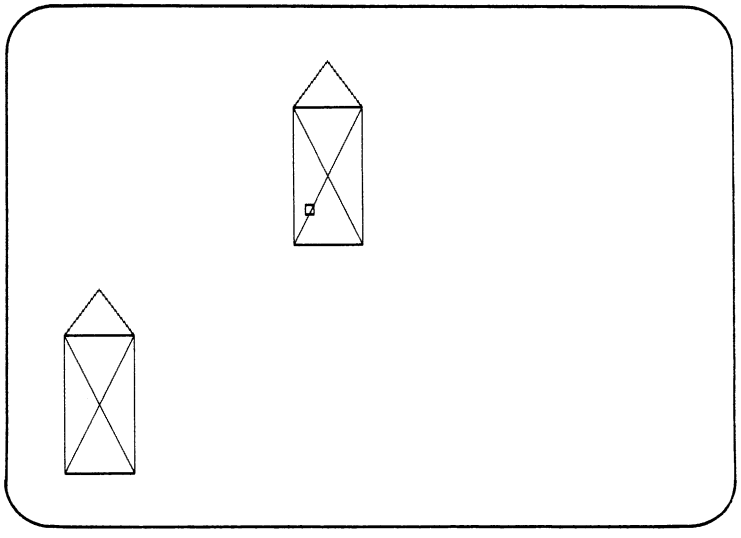
❻ The call to GKS$REQUEST_PICK initiates pick input.

❼ No matter how many times you execute this program, you will never be able to pick HOUSE_2, so that the returned segment name has the value 2. The variable INPUT_STATUS determines whether or not the input is valid (refer to Chapter 8, Input Functions).

If you attempt to pick HOUSE_2, this function always writes an input status value of GKS$K_STATUS_NOPICK and a SEGMENT_NAME value of 1 (the integer value associated with HOUSE_1).

Figure 9–12 shows the screen of a VT241 terminal if the user attempts to pick the undetectable house. Notice that the graphics handler does not outline the house's extent rectangle; this tells the user that the house is not detectable.

Figure 9–12: Setting Pick Detectability—VT241



ZK-5215-86

SET SEGMENT HIGHLIGHTING

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_SEG_HIGHLIGHTING controls the segment attribute that determines whether or not the specified segment is highlighted.

For example, if you use this function to highlight a segment on a VT241, DEC GKS places the segment extent rectangle into an alternative foreground color to draw attention to the specified segment.

If you attempt to highlight an invisible segment, the highlighting does not take effect until you make the segment visible again.

Syntax

GKS$SET_SEG_HIGHLIGHTING *(segment_name, highlighting_flag)*

GSHLIT *(segment_name, high)*

gsethighlight *(segment_name, highlighting)*

Arguments

segment_name

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that identifies a stored segment.

highlighting_flag

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the flag that determines whether or not DEC GKS highlights the specified segment. By default, DEC GKS segments are not highlighted. This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_NORMAL	DEC GKS does not highlight the segment.
1	GKS$K_HIGHLIGHTED	DEC GKS highlights the segment, if visible.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
122	GKS$_ERROR_122	Specified segment does not exist in routine ****

Program Example

Example 9–12 illustrates the use of the function GKS$SET_SEG_ HIGHLIGHTING. Following the program example, Figure 9–13 illustrates the program's effect on a VT241 workstation.

Example 9–12: Highlighting a Segment

```
C     This program draws a house in the lower left corner of the
C     screen and a highlighted house in the upper right.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE_1, HOUSE_2,
     * UPPER_RIGHT_CORNER, NUM_POINTS
      REAL PX ( 9 ), PY ( 9 )
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE_1 / 1 /, HOUSE_2 / 2 /, UPPER_RIGHT_CORNER / 2 /,
     * WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
      CALL GKS$SET_VIEWPORT( UPPER_RIGHT_CORNER, 0.5, 1.0, 0.5, 1.0 )

C     Create a segment in the lower left corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_1 )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Create a second segment in the upper right corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_2 )
      CALL GKS$SELECT_XFORM( UPPER_RIGHT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Highlight HOUSE_2.
      CALL GKS$SET_SEG_HIGHLIGHTING( HOUSE_2, GKS$K_HIGHLIGHTED )

C     Update the surface to initiate the change.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )
```

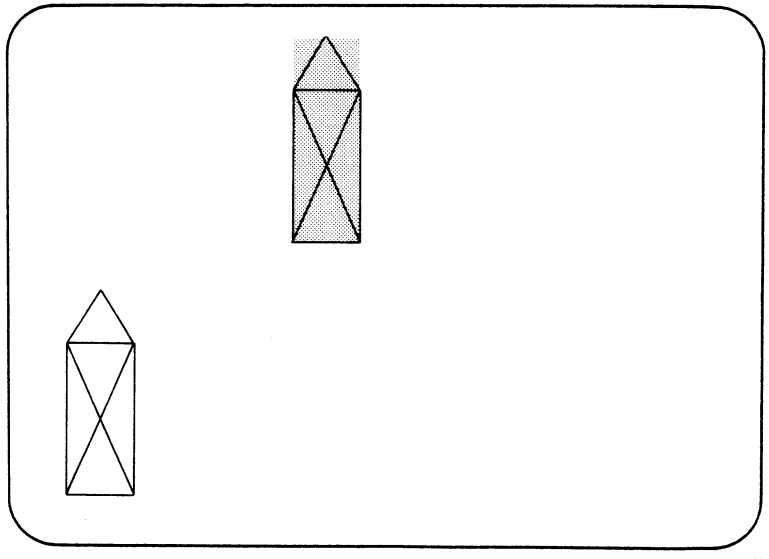
(continued on next page)

Example 9–12 (Cont.): Highlighting a Segment

```
CALL GKS$DEACTIVATE_WS( WS_ID )
CALL GKS$CLOSE_WS( WS_ID )
CALL GKS$CLOSE_GKS()
END
```

Figure 9–13 shows the screen of a VT241 terminal after the first pause in the program.

Figure 9–13: Highlighting a Segment—VT241



SET SEGMENT PRIORITY

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_SEG_PRIORITY sets the segment attribute that determines which segment takes priority when two segments overlap on the workstation surface. The segment priority determines which segment takes precedence on the workstation surface, and which segment is chosen if the user chooses the overlapping area during pick input.

DEC GKS implements segment priority on a scale of real numbers from the value 0.0 to the value 1.0. Segments with the priority 0.0 have the lowest priority, and segments with the priority 1.0 have the highest priority.

Different devices implement segment priority differently. Either a device supports an infinite number of priorities (theoretically), or the device supports a specific number of priorities. If the device supports an infinite amount of priorities, the *maximum number of segment priorities supported* entry in the workstation description table is the value 0. Otherwise, the entry contains the number of priorities supported. (To access this table entry, call the function GKS$INQ_SEG_PRIORITY.)

If the number of priorities supported is not 0.0, then DEC GKS divides the 0.0 to 1.0 priority range into subranges according to the number of supported priorities. If you specify, for two different segments, two different priority values that fall within the same subrange, those segments have the same priority. For instance, if a workstation supports two segment priorities, all segments with the specified values between 0.0 and 0.5 inclusive have the same priority, and values between 0.51 and 1.0 have the same priority.

Syntax

GKS$SET_SEG_PRIORITY *(segment_name, priority)*

GSSGP *(segment_name, priority)*

gsetsegpri *(segment_name, priority)*

SET SEGMENT PRIORITY

Arguments

segment_name

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the integer value that identifies a stored segment.

priority

data type: **real**
access: **read-only**
mechanism: **by reference**

This argument is a real number between the value 0 and the value 1.0 that determines the segment priority. The initial segment priority is the value 0.0. For information concerning your device's implementation of segment priority, refer to the *DEC GKS Device Specifics Reference Manual.*

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
122	GKS$_ERROR_122	Specified segment does not exist in routine ****
126	GKS$_ERROR_126	Segment priority is outside the range [0,1] in routine ****

Program Example

Example 9–13 illustrates the use of the function GKS$SET_SEG_PRIORITY. Following the program example, Figure 9–14 illustrates the program's effect on a VT241 workstation.

Example 9–13: Setting Segment Priorities

```
C     This program draws a house in the lower left corner of the
C     screen and a large house.  Then, the program sets the smaller
C     house to have a higher priority.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, SMALL_HOUSE,
     * LARGE_HOUSE, UNITY, NUM_POINTS, RED
      REAL PX ( 6 ), PY ( 6 ), HIGHER, LOWER
      DATA PX / .4, .1, .1, .25, .4, .4 /
      DATA PY / .1, .1, .7, .9,  .7, .1 /
      DATA NUM_POINTS / 6 /, LOWER_LEFT_CORNER / 1 /,
     * SMALL_HOUSE / 1 /, LARGE_HOUSE / 2 /, UNITY / 0 /,
     * WS_ID / 1 /, HIGHER / 1.0 /, LOWER / 0.0 /, RED / 2 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.7, 0.0, 0.5 )

C     Create a segment in the lower left corner of the surface.
      CALL GKS$CREATE_SEG( SMALL_HOUSE )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
❶     CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_HOLLOW )
      CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Create a second segment in the upper right corner of the surface.
      CALL GKS$CREATE_SEG( LARGE_HOUSE )
      CALL GKS$SELECT_XFORM( UNITY )
❷     CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )
      CALL GKS$SET_FILL_COLOR_INDEX( RED )
      CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()
```

(continued on next page)

Example 9–13 (Cont.): Setting Segment Priorities

```
C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Give the smaller house a higher priority.
❸     CALL GKS$SET_SEG_PRIORITY( LARGE_HOUSE, LOWER )
      CALL GKS$SET_SEG_PRIORITY( SMALL_HOUSE, HIGHER )

C     Redraw the segments.
❹     CALL GKS$REDRAW_SEG_ON_WS( WS_ID )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ These lines of code establish a hollow interior style for the small house.

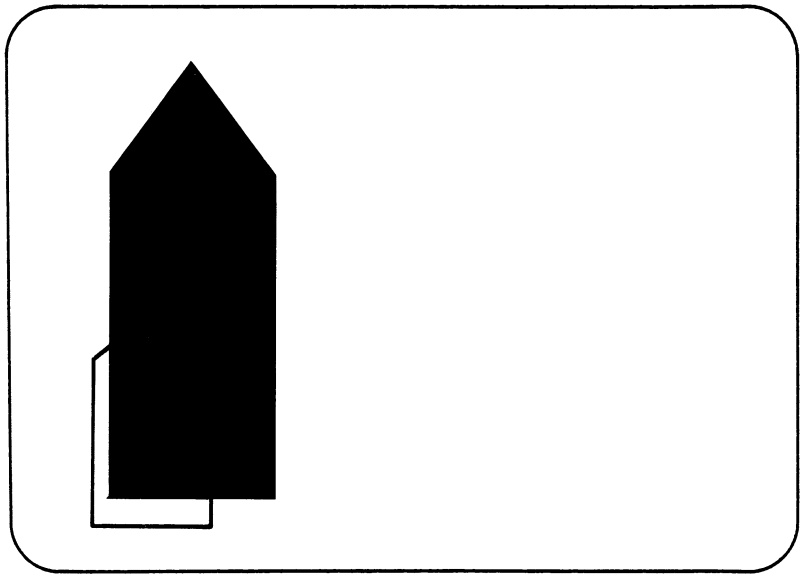
❷ These lines of code establish a solid interior style for the large house. The large house overlaps the smaller house. If you choose the overlapped area during pick input, the device would return the name of the larger house.

❸ By giving the smaller house a priority of 1.0 and the larger house a priority of 0.0, the smaller house has a higher priority.

❹ Once you redraw the segments, the new priorities take effect. The smaller house now overlaps the larger house. If you choose the overlapped area during pick input, the device would now return the name of the smaller house.

Figure 9–14 shows the screen of a VT241 terminal after the program has run to completion.

Figure 9–14: Setting Segment Priorities—VT241



ZK-5100-86

SET SEGMENT VISIBILITY

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_SEG_VISIBILITY sets the segment attribute that determines whether or not a segment is visible on the workstation surface.

A segment must be both visible and detectable if you want the user to be able to choose the segment during pick input.

Syntax

GKS$SET_SEG_VISIBILITY *(segment_name, visibility_flag)*

GSVIS *(segment_name, visible)*

gsetvis *(segment_name, visibility)*

Arguments

segment_name

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that identifies a stored segment.

visibility_flag

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the flag that determines whether or not DEC GKS makes the specified segment visible on the workstation surface. By default, DEC GKS segments are GKS$K_VISIBLE.

Depending on the capabilities of the device, and whether or not the specified segment overlaps other segments, you may need to call either GKS$UPDATE_WS or GKS$REDRAW_SEG_ON_WS to update the picture on the surface of the workstation. For more information, refer to the *DEC GKS Device Specifics Reference Manual.*

This argument can be either of the following values or constants:

Value	Constant	Description
0	GKS$K_INVISIBLE	DEC GKS does not show the segment.
1	GKS$K_VISIBLE	DEC GKS shows the segment on the workstation surface.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
122	GKS$_ERROR_122	Specified segment does not exist in routine ****

Program Example

Example 9-14 illustrates the use of the function GKS$SET_SEG_VISIBILITY.

Example 9–14: Setting the Visibility of a Segment

```
C     This program draws a house in the lower left corner of the
C     screen and an invisible house in the upper right corner.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, LOWER_LEFT_CORNER, HOUSE_1, HOUSE_2,
     * UPPER_RIGHT_CORNER, NUM_POINTS
      REAL PX ( 9 ), PY ( 9 )
      DATA PX / .4, .1, .1, .4, .25, .1, .4, .4, .1 /
      DATA PY / .1, .1, .7, .7, .9,  .7, .1, .7, .1 /
      DATA NUM_POINTS / 9 /, LOWER_LEFT_CORNER / 1 /,
     * HOUSE_1 / 1 /, HOUSE_2 / 2 /, UPPER_RIGHT_CORNER / 2 /,
     * WS_ID / 1 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

      CALL GKS$SET_VIEWPORT( LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5 )
      CALL GKS$SET_VIEWPORT( UPPER_RIGHT_CORNER, 0.5, 1.0, 0.5, 1.0 )

C     Create a segment in the lower left corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_1 )
      CALL GKS$SELECT_XFORM( LOWER_LEFT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Create a second segment in the upper right corner of the surface.
      CALL GKS$CREATE_SEG( HOUSE_2 )
      CALL GKS$SELECT_XFORM( UPPER_RIGHT_CORNER )
      CALL GKS$POLYLINE( NUM_POINTS, PX, PY )
      CALL GKS$CLOSE_SEG()

C     Release deferred output. Pause.  Type RETURN when you are finished
C     viewing the picture.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_POSTPONE_FLAG )
      READ(5,*)

C     Make HOUSE_2 invisible.
      CALL GKS$SET_SEG_VISIBILITY( HOUSE_2, GKS$K_INVISIBLE )
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

C     Pause.  Type RETURN when you are finished
C     viewing the picture.
      READ(5,*)
```

(continued on next page)

Example 9–14 (Cont.): Setting the Visibility of a Segment

```
C     Make HOUSE_2 visible again.
      CALL GKS$SET_SEG_VISIBILITY( HOUSE_2, GKS$K_VISIBLE )

C     Update the surface to initiate the change.
      CALL GKS$UPDATE_WS( WS_ID, GKS$K_PERFORM_FLAG )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

SET SEGMENT TRANSFORMATION

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$SET_SEG_XFORM applies the translation, rotation, and scaling values of a segment transformation to the stored segment. You should use the functions GKS$EVAL_XFORM_MATRIX and GKS$ACCUM_XFORM_MATRIX, described in this chapter, to create the transformation matrix.

DEC GKS applies this segment transformation to all workstations that are currently storing the segment. Changes in segment transformation may or may not take place immediately. Depending on the capabilities of the workstation, you may have to update the surface in order to show the effects of a change in segment transformation (refer to the *DEC GKS Device Specifics Reference Manual.*)

Segment transformations are not cumulative. Every time you call GKS$SET_SEG_XFORM, DEC GKS applies the specified matrix to the segment as stored on NDC space. If you need to have a cumulative effect to segment transformations, refer to GKS$ACCUM_XFORM_MATRIX in this chapter.

To understand the order in which DEC GKS applies different types of transformations, review the transformation and clipping pipeline in Figure 9–7.

Syntax

GKS$SET_SEG_XFORM *(segment_name, transformation_matrix)*

GSSGT *(segment_name, matrix)*

gsetsegtran *(segment_name, segtran)*

Arguments

segment_name

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the integer value that identifies a stored segment.

transformation_matrix

data type: **array (real)**
access: **read-only**
mechanism: **by reference**

This argument is a six-element transformation matrix created previously by a call to either GKS$EVAL_XFORM_MATRIX or GKS$ACCUM_XFORM_MATRIX.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
-22	DECGKS$_ERROR_NEG_22	Invalid segment transformation in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
120	GKS$_ERROR_120	Specified segment name is invalid in routine ****
122	GKS$_ERROR_122	Specified segment does not exist in routine ****

Program Example

Refer to Example 9–2 in this chapter for a program example using a call to GKS$SET_SEG_XFORM.

Chapter 10

Metafile Functions

The DEC GKS metafile functions provide a mechanism for long-term storage, communication, and reproduction of a graphical image. Metafiles created by an application can be used by other applications on other computer systems to reproduce a picture. When you store picture information in a *metafile*, you store specific information concerning the output primitives contained in the picture, the corresponding output attributes, and other information that may be needed to reproduce the picture.

When DEC GKS creates a metafile, it uses one of two formats to store the information about the generated picture. DEC GKS can create either GKSM (GKS Metafiles) or CGM metafiles (Computer Graphics Metafiles).

The format of GKSM metafiles is defined by the GKS standard. When using the GKSM metafile format, DEC GKS stores an audit of the generation of DEC GKS primitives. All of the programs in this chapter create and read GKSM metafiles. For more information concerning GKSM metafiles, refer to Section 10.1.

The format of CGM metafiles is defined by the CGM ANSI X3.122-1986 standard. This metafile format consists of a set of elements that can be used to describe a single graphical picture. CGM metafiles are designed for use with many types of graphics applications, including DEC GKS applications. If you need to create a CGM metafile for use with other applications, possibly on other systems, you can use DEC GKS to create the file. However, DEC GKS cannot read CGM metafiles. For more information concerning CGM metafiles, refer to Section 10.2.

A short-term method of storing output primitives is to store them in segments. For more information concerning segments, refer to Chapter 9, Segment Functions.

10.1 Creating GKSM Metafiles

To create a GKSM metafile, you open and then activate a metafile workstation using the constant GKS$K_GKSM_OUTPUT (numeric value 2) as a workstation type for GKS$K_WSCAT_MO workstations. As the device connection, name the file specification of the file that is to contain the metafile information. (DEC GKS uses the file name exactly as specified, without using a default file extension.) You can open and activate as many GKS$K_GKSM_OUTPUT workstations as determined by the maximum allowable open and active workstations, sending appropriate output to the appropriate active GKS$K_GKSM_OUTPUT workstation.

Once the GKS$K_GKSM_OUTPUT workstation is active, DEC GKS records information about the current state of the picture, such as output attribute information. Then, as you call DEC GKS functions, pertinent information about the function call is recorded in a metafile record. GKS$K_WSCAT_MO workstations record the following information:

- The control functions that affect the appearance of the picture on the workstation surface.
- Output primitives, if the GKS$K_WSCAT_MO workstation is active at the time of the function call. The primitives are stored in a form equivalent to NDC points.
- Output attribute settings that are current at the time of primitive generation.
- Segments, if the GKS$K_WSCAT_MO workstation is active at the time of the call to GKS$CREATE_SEG.
- Geometric attribute data (such as character height, character-up vector, and so forth) affecting stored text primitives, in a form equivalent to normalized device coordinates (NDC).
- Normalization transformation information such as the clipping rectangle. DEC GKS does not record workstation transformations.
- Data that is specific to the application, or information that DEC GKS metafiles cannot store through standard calls to DEC GKS functions (stored using the function GKS$WRITE_ITEM).

If a call to a DEC GKS function is not applicable to the graphical picture, such as calls to certain control or inquiry functions, DEC GKS does not store the function call information in the metafile. Since metafiles record information pertinent to output only, DEC GKS metafiles do not record information about input function calls.

When you create a GKSM metafile, DEC GKS produces a *metafile header*, and for each function call necessary to reproduce the current environment, DEC GKS writes a series of *items* to the metafile. The items generated by a function call roughly correspond to the actual function call or to the state of the picture when the call was made.

For each item, DEC GKS produces an item *header* and an item *data record*. The DEC GKS standard specifies this general format for data storage within GKSM metafiles (metafile header followed by an item header followed by an item data record, and so forth), but the individual item data record format is implementation specific. For instance, some implementations may store all item data as a string of characters, whereas some implementations may store some information as binary-encoded integer values and some information in character strings.

An item *type* is an integer value that corresponds to a DEC GKS function. For instance, an item of type 3 corresponds to a call to GKS$UPDATE_WS. The item type is contained in the item header.

When creating GKSM metafiles, you do not need to be aware of the information contained in the item header or the item data record. Once you activate a GKS$K_GKSM_OUTPUT workstation and call output functions, DEC GKS formats the graphical output information within the metafile for you.

When you close the GKS$K_WSCAT_MO workstation, DEC GKS writes an item of type 0 to the metafile to specify that it is the last item in the metafile.

10.2 Creating CGM Metafiles

To create a CGM metafile, you open and then activate a workstation using the constant GKS$K_CGM_OUTPUT (numeric value 7) as a workstation type for GKS$K_WSCAT_MO workstations. As the device connection, name the file specification of the file that is to contain the metafile information. (DEC GKS uses the file name exactly as specified, without using a default file extension.) You can open and activate as many GKS$K_CGM_OUTPUT workstations as determined by the maximum allowable open and active workstations, sending appropriate output to the appropriate active GKS$K_CGM_OUTPUT workstation.

Once the GKS$K_GKSM_OUTPUT workstation is active, DEC GKS places the graphical information into *elements*, by category. The element categories are as follows:

Category	Description
Delimiter Elements	Separate structures within the metafile.
Metafile Descriptor Elements	Describe the functional content and unique characteristics of the CGM metafile.
Picture Descriptor Elements	Define the limits of the virtual device coordinates (VDCs) and the parameter modes for the attribute elements.
Control Elements	Specify size and and precision of VDC coordinates, and format descriptions of the CGM elements.
Graphical Primitive Elements	Describe the geometric objects in the picture.
Attribute Elements	Describe the various appearances of the graphical elements.
Escape Elements	Describe device- and system-specific functionality.
External Elements	Pass information not needed for the creation of a picture (for instance, a message sent to the user of the metafile).

The elements may have associated data. For instance, the graphical primitive elements may specify VDC points. (The DEC GKS NDC points correspond to the CGM VDC points.) DEC GKS determines the element data from your DEC GKS function calls.

All of the CGM metafile elements are grouped into structures that are similar in appearance to an application program. DEC GKS creates a metafile description at the top of the file. Other structures include the metafile default structure and the metafile picture structure. Each structure begins and ends with the appropriate delimiter elements.

Unlike GKSM metafile items, CGM metafile elements have a certain format, or *encoding*. DEC GKS can create CGM metafile elements in one of the following encodings.

Encoding	Description
Character	This encoding requires that the CGM metafile elements and their parameters be stored in a character-coded format as specified by the CGM standard. Using this encoding, your metafiles use a minimum amount of physical storage.
Binary	This encoding requires that the CGM metafile elements and their parameters be stored in binary code. Using this encoding, many of the applications and machines can store and read CGM metafiles with greater ease. Version 4.0 of DEC GKS does not support binary encoding.
Clear Text	This encoding requires that the CGM metafile elements and their parameters be stored in text. Using this encoding, you can type, print, or edit the CGM metafile so that you can review its contents before reading the file.

To specify an encoding for your metafiles, you can use either of the following bit masks on the command line:

%x00020007 Character encoding

%x00040007 Clear text encoding

If you choose, you can use bitmask constant values within your program to specify an encoding, as follows:

```
      .
      .
      .
      CALL GKS$OPEN_WS( WS_ID, 'CGM_METAFILE.TXT',
     * GKS$K_CGM_OUTPUT .OR. GKS$M_CHARACTER_ENCODING )

C     or,
      .
      .
      .
      CALL GKS$OPEN_WS( WS_ID, 'CGM_METAFILE.TXT',
     * GKS$K_CGM_OUTPUT .OR. GKS$M_CLEAR_TEXT_ENCODING )
      .
      .
      .
```

For more information concerning constants, refer to Appendix B, DEC GKS Constants. For more information concerning bitmasks, refer to Appendix A, DEC GKS Supported Workstations.

Remember that when you create CGM metafiles, you do not need to be aware of the information contained in the individual elements. Once you activate a GKS$K_CGM_OUTPUT workstation and call output functions, DEC GKS formats the graphical output information within the metafile for you.

For detailed information concerning the CGM metafile format for the supported encodings, refer to Appendix E, DEC GKS Metafile Structure.

NOTE

DEC GKS Version 4.0 allows you to create CGM metafiles for other applications that may require this metafile format. This version of DEC GKS cannot read CGM metafiles.

10.3 Reading a GKSM Metafile

To reproduce a graphical image from a GKSM metafile, you must open a metafile input (GKS$K_WSCAT_MI) workstation. DEC GKS defines the constant GKS$K_GKSM_INPUT (numeric value 3) as the workstation type for GKS$K_WSCAT_MI workstations. Also, when you open the GKS$K_GKSM_INPUT workstation, specify the name of the file containing the recorded data items as the connection identifier argument. (DEC GKS uses the file name exactly as specified, without using a default file extension.) You can only open one GKS$K_GKSM_INPUT workstation for every corresponding physical file.

When you open a GKS$K_GKSM_INPUT workstation, the first item that was written to the metafile becomes the *current item*. The current item is the item that is processed when you call the function GKS$GET_ITEM. As with GKS$K_GKSM_INPUT workstations, you can open as many GKS$K_GKSM_INPUT workstations as DEC GKS permits in total workstations, interpreting items from the appropriate metafile on the appropriate active workstations.

To reproduce the graphic image stored in the metafile, you must call GKSGET_ITEM, GKSREAD_ITEM, and GKS$INTERPRET_ITEM for all of the applicable items in the metafile, until you reach the item of type 0 (specifying the last item). The function GKS$GET_ITEM writes the item type, and the length of the data record of the current item, to its last two arguments. The function GKS$READ_ITEM writes the item data record to its last argument and causes the next item in the metafile to become the current item. The function GKS$INTERPRET_ITEM reads information about an item and reproduces the desired action on all active GKS$K_WSCAT_OUTPUT and GKS$K_WSCAT_OUTIN workstations.

In most applications, you call GKS$INTERPRET_ITEM for all items in a metafile. However, there are instances when you may not wish to do this.

For example, if the creator of the metafile called the function GKS$WRITE_ITEM to pass user-defined data to the metafile, then you need to handle this information in a special manner. For instance, if the user-defined data is a text string containing information for the application programmer, then, instead of passing the record to GKS$INTERPRET_ITEM, you should store or write the text string as desired. You can identify user-defined data by checking the item type; all item types greater than 100 are items containing user-defined data. DEC GKS metafiles reserve item data numbers 1 through 100. (If you are not using DEC GKS GKSM metafiles, the reserved item numbers may differ.)

As another example, if you checked the item type and found it to be 3 (which is a call to the function GKS$UPDATE_WS), you may not want to interpret that item if it would delete important output primitives already on the workstation surface. For more information concerning the effects of a call to GKS$UPDATE_WS, refer to Chapter 4, Control Functions.

If after calling GKS$GET_ITEM, you decide that you do not want to interpret the item, pass the value 0 as the data length argument to GKS$READ_ITEM. This skips the current item, causing the next item in the file to become the current item.

10.4 Using the Metafile Functions in Programs

Example 10–1 illustrates the creation of a GKSM metafile.

Example 10–1: Creating a Metafile

```
C     This program creates a metafile that sends information about the
C     programmer who created the metafile, and then draws a filled
C     square in the middle of the NDC space.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, META_OUT, USER_DEFINED, ITEM_LENGTH,
     * NUM_POINTS
      CHARACTER*80 CONTACT_INFO
      REAL PX ( 5 ), PY ( 5 )
      DATA PX / .3, .7, .7, .3, .3 /
      DATA PY / .3, .3, .7, .7, .3 /
      DATA NUM_POINTS / 5 /, WS_ID / 1 /, META_OUT / 2 /,
     * USER_DEFINED / 101 /, ITEM_LENGTH / 80 /
(1)   DATA CONTACT_INFO
     * / 'Programmers: Jim and Cathy D''Augustine.
     * Telephone: 555-5555' /
```

(continued on next page)

Example 10–1 (Cont.): Creating a Metafile

```
      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Open and activate the MO workstation, sending data items to the
C     file METAFILE.DAT.
      CALL GKS$OPEN_WS( META_OUT, 'METAFILE.DAT',
     * GKS$K_GKSM_OUTPUT )
❷     CALL GKS$ACTIVATE_WS( META_OUT )

C     Tell the next user of the metafile whom to call if
C     problems are encountered.
❸     CALL GKS$WRITE_ITEM( META_OUT, USER_DEFINED,
     * ITEM_LENGTH, %REF( CONTACT_INFO ) )

C     Set the interior fill style to solid.
      CALL GKS$SET_FILL_INT_STYLE( GKS$K_INTSTYLE_SOLID )

C     Draw the square in the middle of the NDC space.
      CALL GKS$FILL_AREA( NUM_POINTS, PX, PY )

C     Close and deactivate all workstations
      CALL GKS$DEACTIVATE_WS( META_OUT )
      CALL GKS$CLOSE_WS( META_OUT )
      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ This code intializes user-defined data to be sent to the metafile. This information includes the name of the programmer who created the metafile and the programmer's telephone number.

❷ Once you activate a GKS$K_WSCAT_MO workstation, DEC GKS writes all subsequent output, associated attributes, and current transformation information to the corresponding metafile.

❸ The call to GKS$WRITE_ITEM writes the user-defined data (in this example, information about the programmer who created the metafile) as an item in the metafile. The item type is USER_DEFINED, or, the value 101. Any other programs using the created metafile must know that items numbered 101 pass this type of information; or the application can skip all user-defined information when using the metafile.

For more information, refer to GKS$WRITE_ITEM in this chapter.

This program generates a square, solid, green fill area in the center of the workstation surface.

Example 10–2 reads and interprets the metafile created by Example 10–1.

Example 10–2: Interpreting and Producing a Picture from a Metafile

```
C     This program interprets and produces a picture from a
C     metafile.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
      INTEGER WS_ID, META_IN, USER_DEFINED, ITEM_LENGTH,
     * ITEM_TYPE, MAX_LENGTH, END_METAFILE, CHAR_LENGTH
      REAL ITEM_DATA_RECORD( 500 )
      CHARACTER*80 CONTACT_INFO
      DATA WS_ID / 1 /, META_IN / 2 /,
     * USER_DEFINED / 100 /, MAX_LENGTH / 500 /,
     * END_METAFILE / 0 /, CHAR_LENGTH / 80 /

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )
      CALL GKS$ACTIVATE_WS( WS_ID )

C     Open the MI workstation.
(1)   CALL GKS$OPEN_WS( META_IN, 'METAFILE.DAT',
     * GKS$K_GKSM_INPUT )

(2)   CALL GKS$GET_ITEM( META_IN, ITEM_TYPE, ITEM_LENGTH )
      DO WHILE( ITEM_TYPE .NE. END_METAFILE )

C     If it is the contact programmer information...
(3)   IF ( ITEM_TYPE .GE. USER_DEFINED ) THEN
           CALL GKS$READ_ITEM( META_IN, CHAR_LENGTH,
     *                         %REF( CONTACT_INFO ) )

C     Open a file, store the contact information, and close the file.
           OPEN( UNIT=1, FILE='TEMP.TXT', STATUS='NEW' )
           WRITE(1,*) CONTACT_INFO
           CLOSE( UNIT=1 )

C     Otherwise, read and interpret the item...
(4)   ELSE
           CALL GKS$READ_ITEM( META_IN, MAX_LENGTH,
     *                         ITEM_DATA_RECORD )
           CALL GKS$INTERPRET_ITEM( ITEM_TYPE, ITEM_LENGTH,
     *                            ITEM_DATA_RECORD )
      ENDIF

C     If you want to find out the item types actually written to the
C     metafile, you can include this line.
(5)C  WRITE(6,*) ITEM_TYPE, ITEM_LENGTH
```

(continued on next page)

Example 10–2 (Cont.): Interpreting and Producing a Picture from a Metafile

```
C     Get another item.
      CALL GKS$GET_ITEM( META_IN, ITEM_TYPE, ITEM_LENGTH )
      ENDDO

C     Close and deactivate all workstations
      CALL GKS$CLOSE_WS( META_IN )
      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END
```

The following numbers correspond to the numbers in the previous example:

❶ When you open a GKS$K_WSCAT_MI workstation, you can call GKSGET_ITEM, GKSREAD_ITEM, and GKS$INTERPRET_ITEM. These functions access items stored in the file METAFILE.DAT that was created by the program in Example 10–1.

❷ These lines of code estabish a loop to read all items in the metafile until the program reaches the last item (item of type 0). The example gets a single item and checks to see if it is item 0, which signifies the end of the metafile. If the item is not item 0, then the program enters the loop.

The first call to the function GKS$GET_ITEM is outside of the loop. GKS$GET_ITEM writes the current item's type and size to its last two arguments. The remaining calls to this function occur at the bottom of the loop.

❸ If the item type is greater than the value 100, then the application must handle this user-defined information. This application must know that user-defined data with an item type of 101 contains a character string (specified in Example 10–1).

Once you obtain the string, you can store it in a file, as this program does.

❹ If the item does not contain user-defined data, then interpret the item and send all graphical information to all active workstations.

❺ If you want to see the item types of every item read from this metafile, you can include this line of code. In this example, the item types are the values 71, 61, 1, 101, 43, and so forth. A value of 71 corresponds to the effects of a call to GKS$SET_WS_WINDOW; a value of 61 corresponds to the effects of a call to GKS$SELECT_XFORM; a value of 1 corresponds to the effects of a call to GKS$OPEN_WORKSTATION; a value of 101 corresponds to user-defined data, which cannot be interpreted and must be handled according to its data type; a value of 43 corresponds to the effects of a call to GKS$SET_ASF, and so forth. For a complete list of item types

and their corresponding DEC GKS function calls, refer to Appendix E, DEC GKS Metafile Structure.

If you type the file containing the user-defined data from the metafile, you will see the following:

```
$ TYPE  TEMP.TXT RETURN
Programmers: Jim and Cathy D'Augustine.  Telephone: 555-5555
```

For detailed information concerning the DEC GKS item types and other metafile structural information, refer to Appendix E, DEC GKS Metafile Structure.

10.5 Metafile Inquiries

The following list presents the inquiry functions that you can use to obtain information when writing device-independent code:

GKS$INQ_LEVEL
GKS$INQ_OPEN_WS
GKS$INQ_OPERATING_STATE
GKS$INQ_WS_STATE
GKS$INQ_WSTYPE_LIST

For more information concerning device-independent programming, refer to the *DEC GKS User Manual.*

10.6 Function Descriptions

This section describes the DEC GKS metafile functions in detail. All of the DEC GKS metafile functions work with GKSM metafiles only.

GKS$GET_ITEM

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$GET_ITEM writes the item type and the length of the item data record, from the current item in a metafile, to the last two arguments.

Format

GKS$GET_ITEM *(workstation_id, item_type, item_data_length)*

GGTITM *(workstation_id, item_type, len_dr)*

ggettypegksm *(workstation_id, result)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that represents an open metafile input (GKS$K_WSCAT_MI) workstation. You can open more than one GKS$K_WSCAT_MI workstation at a time, depending on the needs of your application.

item_type

data type:	**integer**
access:	**write-only**
mechanism:	**by reference**

This argument is an integer value that represents the DEC GKS function call corresponding to the metafile item type. For a list of item types and the corresponding function names, refer to Appendix E, DEC GKS Metafile Structure.

item_data_length

data type: **integer**
access: **write-only**
mechanism: **by reference**

This argument is the length of the item data record, in bytes. You should compare this value with your maximum data size to make sure that you defined a data record variable large enough to hold the entire data record.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
34	GKS$_ERROR_34	Specified workstation is not of category MI in routine ****
162	GKS$_ERROR_162	No item is left in GKS Metafile input in routine ****
163	GKS$_ERROR_163	Metafile item is invalid in routine ****

Program Example

For an example of a call to this function, see Example 10–2.

INTERPRET ITEM

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$INTERPRET_ITEM reads an item data record obtained by a call to GKS$READ_ITEM. Then, if the item type corresponds to a call to a function that affects graphical representation, this function makes appropriate changes to the DEC GKS state list, and generates the specified graphical output on all active GKS$K_WSCAT_OUTPUT and GKS$K_WSCAT_OUTIN workstations.

If the item type identifies user-defined data, GKS$INTERPRET_ITEM generates an error indicating that it cannot interpret the item.

Format

GKS$INTERPRET_ITEM *(item_type, item_data_length, item_data_record)*

GIITM *(item_type, len_dr, dim_dr, dr)*

ginterpret *(typeandlenth, date)*

Arguments

item_type

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that represents the DEC GKS function call corresponding to the metafile item type. You can obtain this value by calling the function GKS$GET_ITEM. For a list of item types and the corresponding function names, refer to Appendix E, DEC GKS Metafile Structure.

item_data_length

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the length of the item data record, in bytes. You can obtain this value by calling the function GKS$GET_ITEM.

item_data_record

data type: **record**
access: **read-only**
mechanism: **by reference**

This argument is the item's data record. You can obtain the item's data record by calling the function GKS$READ_ITEM.

Error Messages

Error Number	Completion Status Code	Message
-18	DECGKS$_ERROR_NEG_18	The following error occurred when GKS was interpreting an item, ****
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
161	GKS$_ERROR_161	Item length is invalid in routine ****
163	GKS$_ERROR_163	Metafile item is invalid in routine ****
164	GKS$_ERROR_164	Item type is not a valid GKS item in routine ****

Error Number	Completion Status Code	Message
165	GKS$_ERROR_165	Content of item data record is invalid for the specified item type in routine ****
167	GKS$_ERROR_167	User item cannot be interpreted in routine ****
168	GKS$_ERROR_168	Specified function is not supported in this level of GKS in routine ****

Program Example

For an example of a call to this function, see Example 10–2.

READ ITEM FROM GKSM

Operating States: WSOP, WSAC, SGOP

Description

The function GKS$READ_ITEM reads the current metafile item's data record and then writes the record to its last argument.

You should compare the maximum length for the data record (as passed to this function) with the actual length of the data record (as GKS$GET_ITEM writes to one of its arguments). If the actual size of the record is larger than the maximum allocated space, DEC GKS truncates the record causing loss of information.

After returning the item's data record to the application program, GKS$READ_ITEM makes the next item in the metafile the current item. If you want to skip an item for any reason, specify the value 0 as the maximum record length; this causes the next item in the metafile to become the current item.

Format

GKS$READ_ITEM *(workstation_id, max_record_length, item_data_record)*

GRDITM *(workstation_id, len_dr, len_buf, dr_buf)*

greadgksm *(workstation_id, length, record)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that represents an open metafile input (GKS$K_WSCAT_MI) workstation. You can open more than one GKS$K_WSCAT_MI workstation at a time, depending on the needs of your application.

maximum_record_size

data type: **integer**
access: **read-only**
mechanism: **by reference**

This argument is the maximum length of the declared variable that is to hold the item's data record, in bytes. If the actual data record is larger than this maximum value, DEC GKS truncates the item's data record.

item_data_record

data type: **record**
access: **write-only**
mechanism: **by reference**

This argument is the item's data record.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
7	GKS$_ERROR_7	GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
25	GKS$_ERROR_25	Specified workstation is not open in routine ****
34	GKS$_ERROR_34	Specified workstation is not of category MI in routine ****
162	GKS$_ERROR_162	No item is left in GKS Metafile input in routine ****

Error Number	Completion Status Code	Message
163	GKS$_ERROR_163	Metafile item is invalid in routine ****
165	GKS$_ERROR_165	Content of item data record is invalid for the specified item type in routine ****
166	GKS$_ERROR_166	Maximum item data record is invalid in routine ****

Program Example

For an example of a call to this function, see Example 10–2.

WRITE ITEM TO GKSM

Operating States: WSAC, SGOP

Description

The function GKS$WRITE_ITEM writes a user-defined data item record to a metafile.

For example, you could precede each call to an output function by writing a character string to the metafile, describing the component of the picture generated by the subsequent function call. You can establish a specific item type greater than the value 100 to specify such a description. As an alternative, the application program can treat any item type greater than the value 100 as such a description.

If you are using a metafile structure that is different from a GKSM metafile, you may have to specify different item data record values to this function. For more information concerning the structure of DEC GKS GKSM metafiles, refer to Appendix E, DEC GKS Metafile Structure.

Format

GKS$WRITE_ITEM *(workstation_id, item_type, item_data_length, item_data_record)*

GWITM *(workstation_id, item_type, len_dr, dim_dr, dr)*

gwritegksm *(workstation_id, type, length, data)*

Arguments

workstation_id

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the integer value that represents an active metafile output (GKS$K_WSCAT_MO) workstation. You can activate more than one

GKS$K_WSCAT_MO metafile at one time, depending on the needs of your application.

item_type

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is an integer value that represents the DEC GKS function call corresponding to the metafile item type. You can only use item types greater than the value 100 for user-defined data.

item_data_length

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the length of the item data record, in bytes.

item_data_record

data type:	**record**
access:	**read-only**
mechanism:	**by reference**

This argument is the item's data record.

Error Messages

Error Number	Completion Status Code	Message
-20	DECGKS$_ERROR_NEG_20	GKS not in proper state: GKS in the error state in routine ****
5	GKS$_ERROR_5	GKS not in proper state: GKS shall be either in the state WSAC or in the state SGOP in routine ****
20	GKS$_ERROR_20	Specified workstation identifier is invalid in routine ****
30	GKS$_ERROR_30	Specified workstation is not active in routine ****
32	GKS$_ERROR_32	Specified workstation is not of category MO in routine ****
160	GKS$_ERROR_160	Item type is not allowed for user items in routine ****
161	GKS$_ERROR_161	Item length is invalid in routine ****

Program Example

For an example of a call to this function, see Example 10–1.

Chapter 11

Error-Handling Functions

The DEC GKS error-handling functions provide a method for you to control the generation of messages to the user, and a method of exit when a DEC GKS function call generates an error. The following list presents the DEC GKS error-handling functions:

- GKS$EMERGENCY_CLOSE
- GKS$ERROR_HANDLER
- GKS$LOG_ERROR
- GKS$SET_ERROR_HANDLER

DEC GKS recognizes a number of error situations or conditions. These error conditions are detected within DEC GKS functions, within procedures called by DEC GKS functions (such as calls to the graphics handler procedures), or within other areas of the application program.

For errors occurring in areas of the application program other than in DEC GKS function calls, either the application program regains control or program execution terminates abnormally. If the application program regains control, it can attempt to properly close DEC GKS or, failing that, attempt an emergency closure by calling the function GKS$EMERGENCY_CLOSE. If the program terminates abnormally, the results are unpredictable. In the worst case, you lose all graphical information produced before the error.

For errors detected within procedures called by DEC GKS, if the procedure does not generate a fatal error, then the DEC GKS error handlers may be able to process the error. If the procedure does not generate a fatal error, you should be able to save graphical data. Otherwise, the application program regains control, or the application program is forced to call GKS$EMERGENCY_CLOSE, or in the worst case, you lose all graphical data produced before the error.

For errors detected within DEC GKS functions, DEC GKS performs the following tasks:

1. Sets the DEC GKS error state to ON to prohibit modification of DEC GKS variables.
2. Calls GKS$ERROR_HANDLER and passes the appropriate arguments.
3. Performs function-specific error reaction or cleanup.
4. Sets the DEC GKS error state to OFF.

You can allow DEC GKS to call its own error handler, or you can provide an error handler of your own. An application-supplied, error-handling function can interpret information about the error and store data in a data area for subsequent analysis. Since application-supplied handlers do not have to generate the standard DEC GKS error messages, such a handler can change the format or the text of the messages sent to the user. Also, application-supplied handlers can decide whether to abort a program or to continue despite generated errors, if the errors are not fatal.

A fatal error occurs within DEC GKS when internal data structures are corrupted, or when accurate and meaningful execution of DEC GKS functions is no longer possible. When a fatal error occurs, DEC GKS executes the current error handler and then terminates execution of the application.

The DEC GKS error-handling function calls the error-logging function to display an error message, and then control returns to the error-handling function. The DEC GKS error-handling function GKS$ERROR_HANDLER is available to you as well. You can call GKS$ERROR_HANDLER from an application-supplied handler, if you desire.

The GKS standard dictates that every error-handling function, whether it be the DEC GKS supplied function or an application-supplied function, accept the following information from DEC GKS upon error generation:

- The GKS error number corresponding to the appropriate error condition (refer to Appendix D, DEC GKS Error Messages)
- The name of the GKS function that generated the error condition
- The name of the error file specified in the application program in the call to GKS$OPEN_GKS

To implement an application-supplied, error-handling function, you must define a function with three parameters corresponding to the values listed previously: the DEC GKS error number, the name of the DEC GKS function that generated the error, and the name of the error file. Then, you must pass the address of your error-handling function to GKS$SET_ERROR_HANDLER. For more information, refer to GKS$SET_ERROR_HANDLER in this chapter.

11.1 Function Descriptions

This section describes the DEC GKS error-handling functions in detail. Remember that none of the DEC GKS error-handling functions generate errors.

EMERGENCY CLOSE GKS

Operating States: GKCL, GKOP, WSOP, WSAC, SGOP

Description

The function GKS$EMERGENCY_CLOSE attempts to perform a rapid and orderly closure of DEC GKS.

Usually, you call this function for error conditions detected outside of DEC GKS. If possible, the call to this function closes any open segment, updates all active workstations, deactivates those workstations, closes all open workstations, and then closes DEC GKS.

Syntax

GKS$EMERGENCY_CLOSE

GECLKS

gemergencyclosegks

Program Example

Example 11–1 illustrates the use of the function GKS$EMERGENCY_CLOSE. Following the program example, Figure 11–1 illustrates the program's effect on a VT241 workstation.

Example 11–1: Executing an Emergency Closure of DEC GKS

```
C     This program implements a user-defined error handler.
      IMPLICIT NONE
      INCLUDE 'SYS$LIBRARY:GKSDEFS.FOR'
(1)   INTEGER NEW_ERROR_HANDLER( 2 ), NULL, WS_ID,
     * BAD_WS_ID
      DATA NULL / 0 /, WS_ID / 1 /, BAD_WS_ID / 69 /

(2)   EXTERNAL HANDLE_IT

C     Tell GKS to use this routine instead of GKS$ERROR_HANDLER.
(3)   NEW_ERROR_HANDLER( 1 ) = %LOC( HANDLE_IT )
      NEW_ERROR_HANDLER( 2 ) = NULL

(4)   CALL GKS$SET_ERROR_HANDLER( NEW_ERROR_HANDLER )

      CALL GKS$OPEN_GKS( 'SYS$ERROR:' )
      CALL GKS$OPEN_WS( WS_ID, GKS$K_CONID_DEFAULT, GKS$K_VT240 )

C     Cause an error by passing an identifier of a workstation that
C     isn't open.
      CALL GKS$ACTIVATE_WS( BAD_WS_ID )

      CALL GKS$DEACTIVATE_WS( WS_ID )
      CALL GKS$CLOSE_WS( WS_ID )
      CALL GKS$CLOSE_GKS()
      END


C     HANDLE_IT function.
(5)   FUNCTION HANDLE_IT( ERROR, FUNCTION_NAME,
     * ERROR_FILE_NAME )

      INTEGER ERROR, WS_ID_INVALID, WS_NOT_OPEN
      CHARACTER*80 FUNCTION_NAME, ERROR_FILE_NAME
C     Variables for the corresponding GKS error numbers.
      DATA WS_ID_INVALID / 20 /, WS_NOT_OPEN / 25 /
C     Error decision -- stop if it's a bad workstation.
      IF ( ERROR .EQ. WS_ID_INVALID .OR.
     * ERROR .EQ. WS_NOT_OPEN ) THEN
```

(continued on next page)

Example 11–1 (Cont.): Executing an Emergency Closure of DEC GKS

```
C     Do not continue with the application.
❻        CALL GKS$LOG_ERROR( ERROR, FUNCTION_NAME,
      *                      ERROR_FILE_NAME )
         WRITE(6,*) '---------------------------------'
         WRITE(6,*) 'SEVERE ERROR.'
         WRITE(6,*) '---------------------------------'
         WRITE(6,*) 'After viewing the error message,'
         WRITE(6,*) 'Press RETURN to abort.'

C     Pause.  Type RETURN when you are finished
C     viewing the screen.
         READ(5,*)
         CALL GKS$EMERGENCY_CLOSE()
         STOP

      ELSE

C        Continue normally by logging the error.
❼        CALL GKS$ERROR_HANDLER( ERROR, FUNCTION_NAME,
      *                          ERROR_FILE_NAME )
         RETURN
      END IF
      END
```

The following numbers correspond to the numbers in the previous example:

❶ This code declares the two-element array that you need to pass to GKS$SET_ERROR_HANDLER. For more information, refer to GKS$SET_ERROR_HANDLER in this chapter.

❷ This is the declaration of the external error-handling function.

❸ In the first element of the two-element array passed to GKS$SET_ERROR_HANDLER, you assign the address of the external error-handling function. The second element of the array must always be zero, unless you are programming in Pascal (see the function description in this chapter).

❹ The call to GKS$SET_ERROR_HANDLER tells DEC GKS to pass control to the application's error handler upon error generation. (If you do not implement an application-specified error handler, DEC GKS calls GKS$ERROR_HANDLER upon error generation.)

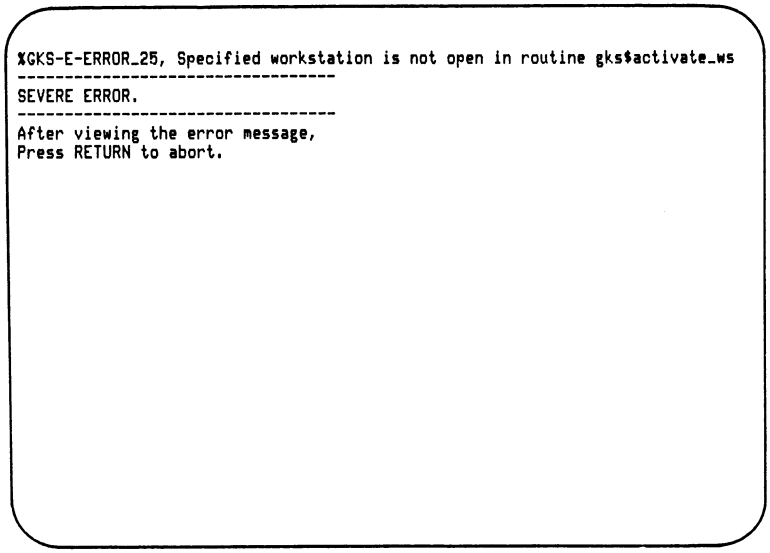
❺ All DEC GKS error handlers must have these parameters: the error number, the function name, and the error file name. As soon as GKS$ACTIVATE_WS encounters the bad workstation identifier in the first program, DEC GKS passes control to this application-defined error handler.

❻ If the error involved a bad workstation identifier (in this example, it did), then this code logs the DEC GKS standard error message and then explains why the program aborts execution. The call to GKS$EMERGENCY_CLOSE attempts an orderly closure of DEC GKS.

❼ If DEC GKS generates any other error, then the call to GKS$ERROR_HANDLER provides the standard DEC GKS response to errors: log the standard error message and continue with execution.

Figure 11–1 shows the screen of a VT241 terminal after the program has run to completion.

Figure 11–1: Executing an Emergency Closure of DEC GKS—VT241



ZK-5103-86

ERROR HANDLING

Operating States: GKCL, GKOP, WSOP, WSAC, SGOP

Description

The function GKS$ERROR_HANDLER calls GKS$LOG_ERROR and allows continued program execution. By default, DEC GKS calls this function when it encounters an error condition.

If you choose, you can write your own error handler to replace this function. If you write your own error handler, you pass the address of your error-handling subroutine to GKS$SET_ERROR_HANDLER (refer to Example 11–1). For more information, refer to GKS$SET_ERROR_HANDLER in this chapter.

For information concerning the various DEC GKS error conditions, refer to Appendix D, DEC GKS Error Messages.

Format

GKS$ERROR_HANDLER *(error_number, function_name, error_file)*

GERHND *(error_number, fun_id, error_file)*

gerrorhand *(error_number, funcname, perrfile)*

Arguments

error_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the number of the DEC GKS error message as dictated by the GKS standard or, in the case of negative error numbers, as dictated by the VMS implementations of GKS. To review the numbers of the DEC GKS error messages, refer to Appendix D, DEC GKS Error Messages.

function_name

data type: **string**
access: **read-only**
mechanism: **by descriptor**

This argument is the text string containing the name of the DEC GKS function that detected the error.

error_file

data type: **string**
access: **read-only**
mechanism: **by descriptor**

This argument is a text string containing the name of the error logging file that was specified in a call to GKS$OPEN_GKS.

Program Example

For an example of a call to this function, see Example 11–1.

LOG ERROR

Operating States: GKCL, GKOP, WSOP, WSAC, SGOP

Description

The function GKS$LOG_ERROR writes the standard DEC GKS error message, which includes the number of the error and the text of the message, to the error file and returns to the procedure or function that called it.

The DEC GKS supplied error handler GKS$ERROR_HANDLER automatically calls this function. An application-supplied error handler can call this function if the need arises.

Format

GKS$LOG_ERROR *(error_number, function_name, error_file)*

GERLOG *(error_number, fun_id, error_file)*

gerrorlog *(error_number, funcname, perrfile)*

Arguments

error_number

data type:	**integer**
access:	**read-only**
mechanism:	**by reference**

This argument is the number of the DEC GKS error message as dictated by the standard or, in the case of negative error numbers, as dictated by the VMS implementations of GKS. To review the numbers of the DEC GKS error messages, refer to Appendix D, DEC GKS Error Messages.

function_name

data type: **string**
access: **read-only**
mechanism: **by descriptor**

This argument is the text string containing the name of the DEC GKS function that detected the error.

error_file

data type: **string**
access: **read-only**
mechanism: **by descriptor**

This argument is a text string containing the name of the error logging file that was specified in a call to GKS$OPEN_GKS.

Program Example

For an example of a call to this function, see Example 11–1.

SET ERROR HANDLER

Operating States: GKCL, GKOP, WSOP, WSAC, SGOP

Description

The function GKS$SET_ERROR_HANDLER establishes an application-defined error handler as the function that DEC GKS calls upon error generation. The error handler, whose address you pass to GKS$SET_ERROR_HANDLER, replaces the DEC GKS supplied error handler GKS$ERROR_HANDLER.

Within your user-defined error handler, you can only call the DEC GKS functions GKS$EMERGENCY_CLOSE, GKS$ERROR_HANDLER, GKS$LOG_ERROR, or any of the inquiry functions.

Format

GKS$SET_ERROR_HANDLER *(new_handler, [old_handler])*

Arguments

new_handler

data type:	**bound procedure value**
access:	**read-only**
mechanism:	**by reference**

This argument is the application-defined, error-handling function or procedure.

This argument is passed by bound procedure value, by reference, for the use of the languages that support up-level addressing (such as Pascal). If your language does not support up-level addressing (in languages such as FORTRAN or BASIC), then you must declare this argument to be an integer array composed of two elements. The first element must contain the address of the application-defined handler, and the second element must contain the value 0. Pass this integer array by reference.

old_handler

data type:	**bound procedure value**
access:	**write-only**
mechanism:	**by reference**

This optional argument is the function or procedure previously used to handle errors. Since this value is 64 bits long, you should declare the argument to be an integer array of two elements, and you pass this array by reference. DEC GKS writes the address of the old error handler to the first element of your array; you can pass this array to GKS$SET_ERROR_HANDLER if at some time you want to reestablish the old error handler.

Program Example

For an example of a call to this function, see Example 11–1.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Reader's Comments

DEC GKS Reference Manual
Volume I
AA–HW43C–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

DEC GKS Reference Manual
Volume I
AA–HW43C–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