DEC GKS

FORTRAN Binding Reference Manual, Part 1

Order Number: AA–MJ29C–TE

June 1992

This manual describes the FORTRAN binding functions provided for DEC GKS™.

Revision/Update Information: This revised manual supersedes the information in the *DEC GKS FORTRAN Binding Reference Manual* (Order No. AA–MJ29B–TE).

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, March 1984
Revised, November 1984, May 1986, March 1987, April 1989, February 1990, June 1992

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

ZK5682

This manual is available on CDROM.

This document was prepared using VAX DOCUMENT, Version 2.1.

Contents

4 Control Functions

5 Output Functions

6 Attribute Functions

8 Segment Functions

9 Input Functions

10 Metafile Functions

Index

Examples

Figures

Tables

Preface

This manual contains complete descriptions for the FORTRAN binding functions provided for DEC GKS. Use this reference material to program DEC GKS on any supported operating system, using any of the languages supported by DEC GKS.

Intended Audience

This manual is for programmers who have experience developing graphics applications in one of the languages supported by DEC GKS. They also should be familiar with the principles of programming DEC GKS, as described in the *DEC GKS User's Guide*.

Structure of This Document

This manual is divided into two parts. Each chapter deals with a specific subject or group of functions, describing the syntax and arguments for each function. The appendixes provide additional information you may find useful. Part 1 includes the following chapters:

- Chapter 1 provides an introduction to DEC GKS.
- Chapter 2 provides information about DEC GKS and the VMS™ operating system.
- Chapter 3 provides information about DEC GKS and the ULTRIX™ operating system.
- Chapter 4 describes the functions you use to control DEC GKS and workstation environments.
- Chapter 5 describes the functions you use to generate output primitives.
- Chapter 6 describes the functions you use to generate attributes.
- Chapter 7 describes the functions you use to set up and perform normalization and workstation transformations.
- Chapter 8 describes the functions you use to store output primitives in segments.
- Chapter 9 describes the functions you use to accept input from workstations.
- Chapter 10 describes the functions you use to store graphic images as metafiles.

Part 2 includes the following chapters and appendixes:

- Chapter 11 describes the functions you use to inquire for information about the capabilities and state of the DEC GKS system.
- Chapter 12 describes the functions you use to handle errors.

- Appendix A lists DEC GKS error codes, along with the corresponding severity code and message for each one.
- Appendix B lists constants defined for the FORTRAN binding interface.
- Appendix C provides a list of code examples available throughout this manual, listed alphabetically, by function.
- Appendix D lists the DEC GKS functions and the corresponding FORTRAN binding names.
- Appendix E lists specific input values that apply to all DEC GKS workstations that perform both input and output.
- Appendix F provides implementation-specific information about DEC GKS.

Associated Documents

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

- *DEC GKS User's Guide*—for programmers who need information that supplements the DEC GKS binding manuals
- *DEC GKS GKS$ Binding Reference Manual*—for programmers who need specific syntax and argument descriptions for the GKS$ binding
- *DEC GKS GKS3D$ Binding Reference Manual*—for programmers who need specific syntax and argument descriptions for the GKS3D$ binding
- *DEC GKS C Binding Reference Manual*—for programmers who need specific syntax and argument descriptions for the C binding
- *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*—for programmers who need information about specific devices
- *Building a Device Handler System for DEC GKS and DEC PHIGS*—for programmers who need to build workstation graphics handlers

Conventions

The following conventions are used in this manual:

Convention	Meaning
RETURN	The symbol RETURN represents a single stroke of the Return key on a terminal.
Boldface text	Boldface text represents the introduction of a new term. In interactive examples, user input appears in boldface type.
Italic text	Italic text indicates a parameter name or a book name. DEC GKS description table and state list entry names, and workstation description tables and state list entry names are also italicized.
UPPERCASE TEXT	Uppercase text indicates a DEC GKS function or symbol name.
. . .	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.
. . .	A horizontal ellipsis indicates that additional arguments, options, or values can be entered. A comma preceding the ellipsis indicates that successive items must be separated by commas.
	Horizontal ellipses in illustrations indicate that there is information not illustrated that either precedes or follows the information included in the illustration itself.
[]	Square brackets, in function synopses and a few other contexts, indicate that a syntactic element is optional.

Introduction

Insert tabbed divider here. Then discard this sheet.

1
Introduction to DEC GKS

DEC GKS is a development tool that creates two- and three-dimensional graphics applications that are system and device independent. It is Digital's level 2c implementation, compliant with the Graphical Kernel System (GKS) defined by the American Standards Institute (ANSI X3.124–1985), the International Standard (ISO/IS 7942), and the Graphical Kernel System for Three Dimensions (GKS–3D) defined by the International Standard (ISO/IS 8805). The DEC GKS FORTRAN binding conforms to the ISO GKS–3D Draft International Standard ISO/IEC/DIS 8806/1 and the GKS American National Standards Institute (ANSI X3.124.1–1985).

DEC GKS is a system- and device-independent graphics library that enables the development of GKS applications that can be moved to other platforms (hardware devices or operating systems) or that generate output on other graphic devices without modification to the source code. It provides functionality such as output primitives, logical workstation management, workstation-dependent and workstation-independent segment storage, six types of logical input devices, synchronous and asynchronous input, inquiries returning the system's capabilities, and metafile input and output.

DEC GKS implements syntactical language bindings. For DEC GKS, these include the DEC GKS FORTRAN and DEC GKS C bindings. The language bindings in general, and specifically the FORTRAN binding, 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. Digital recommends that you use the C or FORTRAN language bindings, because you will have better portability and ease of use.

DEC GKS also implements the functional standard using function names beginning with the prefixes GKS$ and GKS3D$. If you use the GKS$ or GKS3D$ functions, you have to edit your program if you want to transport the program across systems or across GKS implementations.

1.1 GKS Function Categories

The DEC GKS function categories are as follows:

- Control
- Output
- Attribute
- Transformation
- Segment
- Input
- Metafile

- Inquiry
- Error-Handling

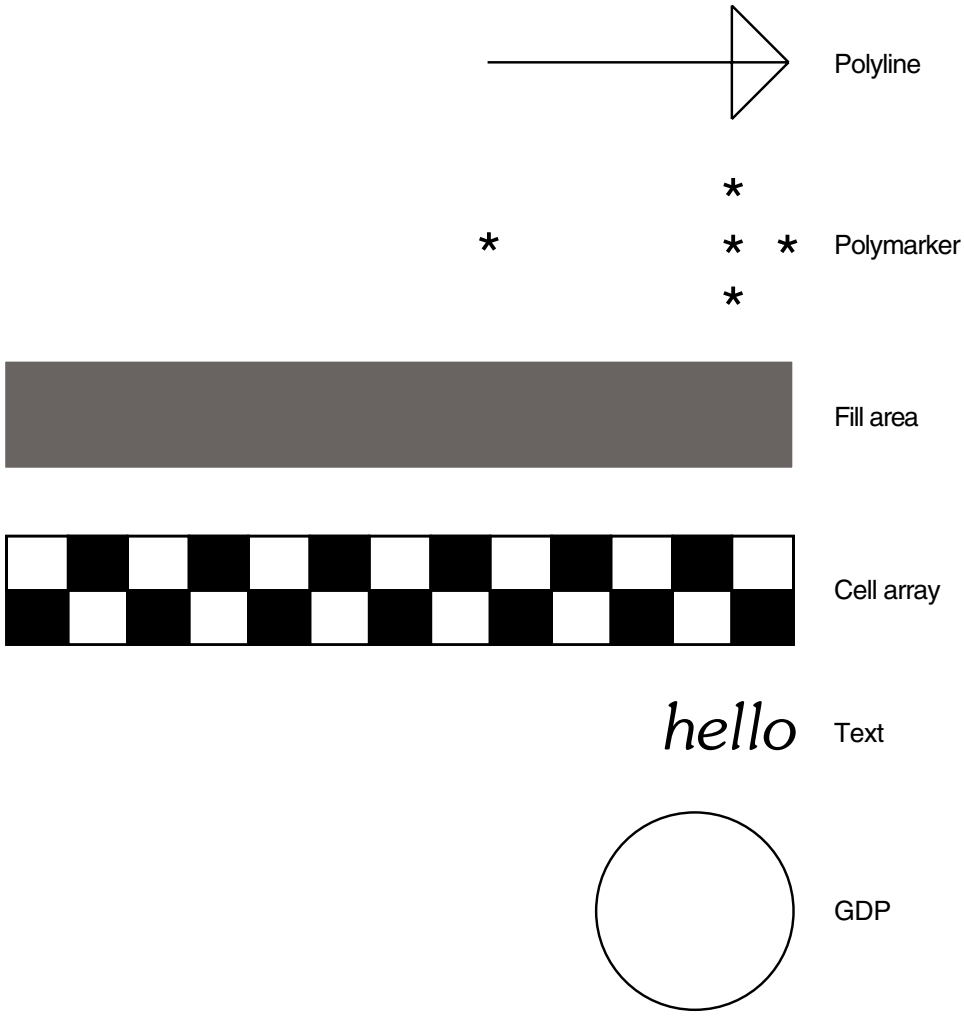
The control functions determine which DEC GKS functions you can call at a given point in your program. They also control the buffering of output and the regeneration of segments on the workstation surface.

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 DEC GKS Output Primitives



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

Transformations affect the composition of the graphic picture and the presentation of that picture. There are **normalization**, **workstation**, and **viewing** 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 into a picture, or zoom out of a picture.

The viewing transformations control the orientation and projection of the picture.

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

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

The metafile functions allow you to store and 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, see Chapter 10, Metafile Functions.

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

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, see Chapter 12.

If you need more tutorial information concerning DEC GKS concepts, see the *DEC GKS User's Guide*.

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 the GKS output capabilities (level 2) and all 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.

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, see Chapter 9, 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, see Chapter 8, Segment Functions.

1.3 Function Presentation Format

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

1.3.1 Function Header

Each function header in this manual includes 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.

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–GE

1.3.2 Function Operating States

The operating states section lists the valid operating states during which a call to the function is permitted (for more information, see Chapter 4, Control Functions).

1.3.3 Function Syntax

The syntax section lists the syntax of a call to the DEC GKS function. This syntax includes the argument list. Each argument is described in the Syntax section.

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

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier

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

For each argument, the specified values include:

- The data type of the argument
- The type of access made by the argument

Most of the passing mechanisms required by DEC GKS functions are the default mechanisms of VAX FORTRAN™.

Some of the data type descriptions in this manual are not worded in exactly the same manner as in the VMS™ documentation. For example, 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 description of the argument-passing mechanisms, see the *VMS Run-Time Library Routines Reference Manual*. For information concerning language-specific passing extensions, see the appropriate VAX high-level language manual.

All the DEC GKS functions always return an integer condition status value. For the FORTRAN language, the status value is only accessible with a nonstandard extension of the FORTRAN binding.

1.3.4 Constants

The constants section lists the DEC GKS constants that are defined for each enumerated type and a description of each, in order of appearance in the function call. For a complete list of the DEC GKS constants, see Appendix B.

1.3.5 Function Description

The description section describes the function in detail. The description contains pertinent information about the DEC GKS operating state, the GKS description table and state list, and the workstation description table and state list.

1.3.6 See Also Section

Most of the functions include a See Also section. This section lists related functions and gives pointers to code examples, located at the end of each chapter, that include the specified function.

Program Examples Section

Appendix C lists all the functions called in the code examples. The program examples are also available on line. They are located in GKS$EXAMPLES on VMS systems, and in /usr/lib/GKS/examples on ULTRIX systems.

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

```
         .
         .
         .
C     Release the deferred output and pause the display for "timeout"
C     seconds.

      CALL GUWK( WS_ID, GPOSTP )
      CALL GWAIT( TIMEOUT, WS_ID, ICL, DEVNUM )
         .
         .
         .
```

Because DEC GKS allows workstations to **defer**, or buffer, output, you have to update the screen with a call to UPDATE WORKSTATION to view the picture created by all previous function calls in the program. The call to the AWAIT EVENT function causes program execution to pause.

Considering that the rate of deferral may differ on various workstations, you may wish to use the function INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES to check the current deferral mode. If the deferral mode is anything other than ASAP, you may wish to update the workstation surface periodically when you are debugging your program. If you want to change the deferral mode so the workstation surface is always current, you can call the function SET DEFERRAL STATE to change the current deferral mode to ASAP.

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

Also, most program examples include the following lines of code:

```
            .
            .
            .
      WS_ID = 1

      CALL GOPWK( WS_ID, GCONID, GWSDEF )
            .
            .
            .
```

This code tells DEC GKS to use the default values for the connection identifier and the workstation type. The default values are defined by the logical names GKS$CONID and GKS$WSTYPE on VMS systems, and by the environment variables GKSconid and GKSwstype on ULTRIX systems. To change the default values of GKS$CONID and GKS$WSTYPE on a VMS system, enter the following commands:

```
$ DEFINE GKS$CONID FOOBAR::0 RETURN
$ DEFINE GKS$WSTYPE 211 RETURN
```

After entering these commands, the new value for GKS$CONID is the FOOBAR node, and the new value for GKS$WSTYPE is 211 (DECwindows™ XUI workstation).

To change the values of GKSconid and GKSwstype on an ULTRIX system, enter the following commands:

```
# setenv GKSconid FOOBAR::0 RETURN
# setenv GKSwstype 211 RETURN
```

After entering these commands, the new value for GKSconid is the FOOBAR node, and the new value for GKSwstype is 211 (DECwindows XUI workstation). For more information on specifying environment options on VMS and ULTRIX operating systems, see Chapter 2 and Chapter 3.

Following many of the program examples, there is an illustration representing the graphic image generated on the surface of the DECwindows XUI workstation. Because 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 example, the lines may not be as perfectly smooth as presented in the figure. The figures in this manual serve the purpose of showing relative positioning and general shape of the graphic image on the surface of a DECwindows XUI workstation.

VMS Programming

Insert tabbed divider here. Then discard this sheet.

2
VMS Programming Considerations

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

The information contained in this chapter was correct when the manual went to press. However, the information may have been changed. For the most up-to-date information on using DEC GKS on VMS systems, see the following files:

SYS$HELP:DECGKS_FBIND_OP_SPEC.PS
SYS$HELP:DECGKS_FBIND_OP_SPEC.TXT

2.1 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 programming language you use.

DEC GKS constants and their values must be made available to all programs that call DEC GKS functions, regardless of the programming language you use. All high-level programming 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 includes 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 appropriate for the programming language you are using.

FORTRAN provides the INCLUDE statement for inserting an external file into a program. Therefore, any FORTRAN program that uses the DEC GKS FORTRAN binding should contain the following line:

```
INCLUDE  'SYS$LIBRARY:gks.f'
```

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 file located in SYS$LIBRARY is gks.f.

An alternate method of including the gks.f file is to copy it to your local directory and use the statement `INCLUDE 'gks.f'` in the application program. This method is used in the example programs in this manual.

2.2 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 appropriate for the programming language you are using, and use the LINK and RUN commands to link the object file and execute the program image.

DEC GKS functions are supplied as an installed shareable image library, which makes linking faster and easier, makes the resulting executable image file smaller, and allows your application to be upgraded with new versions of DEC GKS without having to be rebuilt.

On VMS systems, a DEC GKS program can be linked with the following DCL command:

```
$ LINK program, SYS$LIBRARY:GKS$FORBND/LIB RETURN
```

2.3 Opening a Workstation

The following sections contain information on specifying the workstation connection identifier and workstation type.

2.3.1 Specifying the Connection Identifier

An application can specify the connection to a device by passing the connection identifier (ID) to the OPEN WORKSTATION function using a logical unit number (LUN). The LUN can be used in any of the following ways:

- Explicitly: Use a FORTRAN OPEN function to associate a LUN with a device. Pass the LUN to the OPEN WORKSTATION call.
- Default: Pass a LUN of 0. DEC GKS now attempts to translate the logical name GKS$CONID. If no translation exists, DEC GKS uses GKS_DEFAULT.OUTPUT, which specifies a file in the current directory as the connected device. The user can define the VMS logical name GKS$CONID.
- Escape: Use the Set Connection Identifier String escape (–440). For more information on this escape, see the OPEN WORKSTATION and ESCAPE functions, and the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

2.3.2 Specifying the Workstation Type

The application can specify the workstation type to the OPEN WORKSTATION function in either of the following ways:

- Use the DEC GKS workstation types. Pass any of the workstation types defined in the language file SYS$LIBRARY:gks.f.
- Use the default workstation type by passing a value of 0. DEC GKS attempts to translate the VMS logical name GKS$WSTYPE. If no translation exists, DEC GKS uses the workstation type 35 (LA75™ printer). The user can define the VMS logical name GKS$WSTYPE.

2.4 DEC GKS Logical Names

Within DEC GKS there are a number of logical names that are interpreted at run time. These logical names allow a specific application (or system) to tailor DEC GKS to best suit the needs of the application or device. Each of the logical names controls some aspect of the overall run-time environment of the DEC GKS session. All the logical names have to be set before starting a GKS session and remain constant during a session. Altering logical names during a session has no effect on the logicals.

2.5 Defining Logical Names

On VMS systems, the logical names are defined at DCL level as follows:

```
$ define GKS$logical value
```

Logical names and values can be either uppercase or lowercase strings.

DEC GKS searches for VMS logical names in three different locations. Once the logicals are found, the search stops. The locations are searched in the following order:

1. The PROCESS logical table.
2. The GROUP logical table.
3. The SYSTEM logical table. This is the default location to define logical names.

2.6 Types of Logical Names

The DEC GKS logical names can be divided into two groups:

- General DEC GKS logical names
- Graphics-handler logical names

The following section describes the general logical names available with DEC GKS. For information on the graphics-handler logical names, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

2.6.1 General Logical Names

Table 2–1 lists the general logical names available with DEC GKS.

Table 2–1 General Logical Names for DEC GKS

Logical Name	Value	Description
GKS$ASF	INDIVIDUAL or BUNDLED	Specifies the default aspect source flag (ASF) setting to be either BUNDLED or INDIVIDUAL. The predefined value is INDIVIDUAL.
GKS$CONID	String containing any valid workstation connection identifier	Specifies the default workstation connection identifier to be used in the call to OPEN WORKSTATION, if the caller passes CONID 0. The predefined setting is platform-dependent.

(continued on next page)

Table 2–1 (Cont.) General Logical Names for DEC GKS

Logical Name	Value	Description
GKS$DEF_MODE	ASAP, BNIG, BNIL or ASTI	Specifies the default deferral mode to be ASAP (as soon as possible), BNIG (before the next interaction globally), BNIL (before the next interaction locally), or ASTI (at some time). The predefined value is defined in the workstation description table.
GKS$ERRFILE	String containing any valid error file specification	Specifies the default error file to be used in the OPEN GKS function, if the user passes a NIL (0) error file descriptor. On VMS systems, this is set to SYS$ERROR: by default.
GKS$ERROR	ON or OFF	Specifies whether the default standard error checking is ON or OFF. The default value is ON. Note that if you turn error checking OFF, you may improve overall DEC GKS throughput, but DEC GKS may terminate in an uncontrolled way.
GKS$IRG	SUPPRESSED or ALLOWED	Specifies the default implicit regeneration mode (IRG) to be set to either SUPPRESSED or ALLOWED. The predefined value is defined in the workstation description table.
GKS$METAFILE_TYPE	GKSM or GKS3	Specifies the dimension of the metafile output. The value GKSM is for two-dimensional metafile output; GKS3 is for three-dimensional metafile output. The default value is GKS3.
GKS$NDC_CLIP	ON or OFF	Specifies the default normalized device coordinate (NDC) clipping to ON or OFF. The predefined value is ON.
GKS$STROKE_FONT1	String containing the file path for stroke font1	Specifies the default stroke font 1 to be used.
GKS$WSTYPE	String containing any valid workstation type number	Specifies the default workstation type to be used in the call to OPEN WORKSTATION, if the caller passes wstype 0. The predefined setting is platform-dependent.

2.7 Error Handling

The following sections contain information on error codes and error files.

2.7.1 Error Codes

Each DEC GKS function call returns a DEC GKS error value to the calling routine. All the returned error values are defined in the language file gks.f.

The function call can include a check of the returned status. The value 0 is returned if the function has executed successfully. If this value is not returned, the function has failed to execute successfully. See Appendix A for more information about DEC GKS errors codes. For the FORTRAN language, the error value is only accessible with a nonstandard extension of the DEC FORTRAN binding.

2.7.2 Error Files

Error messages are normally written to an error logging file. The application can specify the file name to the OPEN GKS function by using a logical unit number (LUN). The LUN can be used in any of the following ways:

- Explicitly: Use a FORTRAN OPEN function to associate a LUN with a device. Pass the LUN to the OPEN GKS call.
- Implicitly: Pass a LUN in the range 1 to 255, without explicitly opening an error file. DEC GKS writes error messages to a file named FORT*nnn*.DAT, where *nnn* is the 3-digit representation of the LUN.
- Default: Pass a LUN of 0.

  DEC GKS opens the default error file. To do this, it first attempts to translate GKS$ERRFILE. If no translation exists, DEC GKS uses SYS$ERROR as the file pointer. The user can define the VMS logical name GKS$ERRFILE.

If no messages have been written to the error file, a call to CLOSE GKS deletes the file.

ULTRIX Programming

Insert tabbed divider here. Then discard this sheet.

3
ULTRIX Programming Considerations

The specific method for using DEC GKS software depends on the features and conventions of each programming language. This section describes general issues that must be considered when using the FORTRAN–77 (f77) compiler with DEC GKS.

The information contained in this chapter was correct when the manual went to press. However, the information may have been changed. For the most up-to-date information on using DEC GKS on ULTRIX systems, see the following files:

/usr/lib/GKS/doc/decgks_fbind_op_spec.ps
/usr/lib/GKS/doc/decgks_fbind_op_spec.txt

3.1 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 programming language you use.

DEC GKS constants and their values must be made available to all programs that call DEC GKS functions, regardless of the programming language you use. Incorporating an external file is the method for making DEC GKS constants available.

Your installation kit includes several files that contain DEC GKS constants and separate files that contain DEC GKS completion status code constants.

The language definition file for the FORTRAN binding is /usr/include/GKS/gks.f.

You incorporate these files into your FORTRAN program with the statement:

```
INCLUDE /usr/include/GKS/gks.f
```

An alternate method of including the gks.f file is to copy it to your local directory and use the statement `INCLUDE 'gks.f'` in the application program. This method is used in the example programs in this manual.

3.2 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 appropriate for the processor you are using. To run an executable program, type the executable file name that you specified. The following sections describe how to compile, link, and run your programs on ULTRIX systems with RISC processors.

3.2.1 Linking DEC FORTRAN Programs on ULTRIX Systems with RISC Processors

To compile and link a DEC GKS program using DEC FORTRAN™ on ULTRIX systems with RISC processors using all device handlers except the DECwindows XUI device handler, use the following command:

```
# f77 -fpe1 -o program program.f [gksconfig.o] \ RETURN
-lGKSdforbnd -lGKS /usr/lib/DXM/lib/Mrm/libMrm.a \ RETURN
/usr/lib/DXM/lib/Xm/libXm.a /usr/lib/DXM/lib/Xt/libXt.a \ RETURN
-lddif -lcursesX -li -lc -lX11 -llmf RETURN
```

To compile and link a DEC GKS program using DEC FORTRAN on ULTRIX systems with RISC processors, using the DECwindows XUI device handler, use the following command:

```
# f77 -fpe1 -o program program.f gks_decw_config.o -lGKSdforbnd \ RETURN
-lGKS -lddif  -ldwt -lcursesX -li -lc -lX11 -llmf RETURN
```

The gksconfig.o file is optional when using any device handler except the DECwindows XUI device handler. The file gks_decw_config.o is required by the DECwindows XUI device handler. The -fpe1 switch prevents floating point exception run-time errors when you use DEC GKS text facilities.

You must build the gksconfig.o and gks_decw_config.o files. To build the object files, use the following command:

```
# cc -c configuration_file.c RETURN
```

Replace *configuration_file.c* with either gksconfig.c or gks_decw_config.c.

A workstation or device handler can be deliberately excluded from the executable image to minimize image size and link time. This can be done by customizing the configuration file and specifying the customized version in the `link` command. The installed version of the configuration file is /usr/lib/GKS/gksconfig.c.

There is also an installed version of the configuration file that must be used when linking with the DECwindows XUI device handler. It is /usr/lib/GKS/gks_decw_config.c. See Section 3.9 for more information on how to use configuration files.

The options to the link command are required if certain default handlers are included in the configuration file, as follows:

- The switches [-lc] and [-lx11] are required for the DECwindows XUI and Motif® device handlers.
- The switch [-ldwt] is required for the DECwindows XUI device handler.
- The switch [-lddif] is required for the DDIF™ device handler.
- The library [-lcursesX] is required for any device handler for workstations capable of input, output, or both.
- The following libraries are required for the Motif device handler:

  /usr/lib/DXM/lib/Mrm/libMrm.a
  /usr/lib/DXM/lib/Xm/libXm.a
  /usr/lib/DXM/lib/Xt/libXt.a

3.2.2 Linking FORTRAN Programs on ULTRIX Systems with RISC Processors

To compile and link a DEC GKS FORTRAN program on ULTRIX systems with RISC processors using all device handlers except the DECwindows XUI device handler, use the following command:

```
# f77 -o program program.f [gksconfig.o] -lGKSforbnd \ RETURN
-lGKS /usr/lib/DXM/lib/Mrm/libMrm.a /usr/lib/DXM/lib/Xm/libXm.a \ RETURN
/usr/lib/DXM/lib/Xt/libXt.a \ RETURN
-lddif -lcursesX -lc -lX11 -llmf -lm RETURN
```

To compile and link a DEC GKS FORTRAN program on ULTRIX systems with RISC processors using the DECwindows XUI device handler, use the following command:

```
# f77 -o program program.f gks_decw_config.o -lGKSforbnd \ RETURN
-lGKS -lddif -ldwt -lcursesX -lc -lX11 -llmf -lm RETURN
```

The gksconfig.o file is optional. The file gks_decw_config.o is required by the DECwindows XUI device handler.

You must build the gksconfig.o and gks_decw_config.o files. To build the object files, use the following command:

```
# cc -c configuration_file.c RETURN
```

Replace *configuration_file.c* with either gksconfig.c or gks_decw_config.c.

A workstation or device handler can be deliberately excluded from the executable image to minimize image size and link time. This can be done by customizing the configuration file and specifying the customized version in the `link` command. The installed version of the configuration file is /usr/lib/GKS/gksconfig.c.

There is also an installed version of the configuration file that must be used when linking with the DECwindows XUI device handler. It is /usr/lib/GKS/gks_decw_config.c. See Section 3.9 for more information on how to use configuration files.

The options to the link command are required if certain default handlers are included in the configuration file, as follows:

- The switches [-lc] and [-lx11] are required for the DECwindows XUI and Motif device handlers.
- The switch [-ldwt] is required for the DECwindows XUI device handler.
- The switch [-lddif] is required for the DDIF™ device handler.
- The library [-lcursesX] is required for any device handler for workstations capable of input, output, or both.
- The following libraries are required for the Motif device handler:

  /usr/lib/DXM/lib/Mrm/libMrm.a
  /usr/lib/DXM/lib/Xm/libXm.a
  /usr/lib/DXM/lib/Xt/libXt.a

3.3 Opening a Workstation

The following sections contain information on specifying the workstation connection identifier and workstation type.

3.3.1 Specifying the Connection Identifier

The application can specify the connection by passing the connection ID to the OPEN WORKSTATION function in any of the following ways:

- Explicitly: Use a FORTRAN OPEN function to associate a LUN with a device. Pass the LUN to the OPEN WORKSTATION call.
- Default: Pass a LUN of 0. DEC GKS now attempts to translate the environment variable GKSconid. If no translation exists, DEC GKS uses gks_default.output.
- Escape: Use the Set Connection Identifier String escape (–440). For more information on this escape, see the OPEN WORKSTATION and ESCAPE functions, and the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

3.3.2 Specifying the Workstation Type

The application can specify the workstation type to the OPEN WORKSTATION function in either of the following ways:

- Use the DEC GKS workstation types. Pass any of the workstation types defined in the file /usr/include/GKS/gks.f.
- Use the default workstation type by passing a value of 0. DEC GKS attempts to translate the environment variable GKSwstype. If no translation exists, DEC GKS uses the workstation type 35 (LA75 printer).

3.4 DEC GKS Environment Variables

Within DEC GKS there are a number of environment variables that are interpreted at run time. These environment variables allow a specific application (or system) to tailor DEC GKS to best suit the needs of the application or device. Each of the environment variables controls some aspect of the overall run-time environment of the DEC GKS session. All the environment variables have to be set before starting a GKS session, and remain constant during a session. Altering environment variables during a session has no effect on the value of the environment variable.

3.5 Defining Environment Variables

On ULTRIX systems, the environment variables are defined in a file named .GKSdefaults in the user's login directory, or in the system file /usr/lib/GKS/.GKSdefaults.

The following examples show the syntax you use to define environment variables in the .GKSdefaults file:

```
GKSconid          :  gks_default.output    # connection id, device, or file name
GKSwstype         :  35                    # workstation type (LA 75)
```

The environment variables can also be defined at the csh or sh level. To define an environment variable at csh level, use the following syntax:

```
# setenv GKSenvironment_variable value
```

To define an environment variable at sh level, use the following syntax:

```
# GKSenvironment_variable=value
```

The values you assign to environment variables can be either uppercase or lowercase strings. However, the environment variable names are case sensitive.

DEC GKS searches for the environment variables in three different locations. Once the environment variables are found, the search stops. The locations are searched in the following order:

1. User-specific ULTRIX environment variables.
2. User-specific environment variables defined in the file ~/.GKSdefaults. Digital recommends that you define the environment variables in this file.
3. System-wide environment variables defined in the file /usr/lib/GKS/.GKSdefaults.

3.6 The Default Environment Variable File

The default environment variable file, .GKSdefaults, contains the following:

```
!
! GKS Default Settings
!
GKSconid          :  gks_default.output    # connection id, device, or file name
GKSwstype         :  35                    # workstation type (LA 75)
GKSerror          :  on                    # on or off
GKSasf            :  individual            # bundled or individual
GKSndc_clip       :  on                    # on or off
GKSerrfile        :  stderr                # device or file name
GKSmetafile_type  :  gks3
GKSstroke_font1   :  /usr/lib/GFX/font/gfx_font_neg1 # Stroke font 1
!
! file : /usr/lib/GKS/.GKSdefaults
!
! GKS System-Wide Environment Definitions
! ========================================
!
! Environment variables allow you to customize the DEC GKS environment
! to suit your needs.
!
! i)  Modify the GKS system wide default settings.
!
! Edit this file to change GKS system-wide environment variable
! default settings.
!
! ii) Modify the GKS user-specific environment variable default settings
!     using the ~/.GKSdefaults file
!
! Copy this file into your login account i.e. ~/.GKSdefaults and
!       modify it to suit your needs.
!
! iii) Modify the GKS user-specific settings using ULTRIX
!      environment variables.
!
!      For example : setenv GKSwstype 10
!
!
!
! The DEC GKS search order for environment variables translation is :
! ------------------------------------------------------------------
!
! 1) User-specific ULTRIX environment variable
!
! 2) User-specific environment variables defined in the file ~/.GKSdefaults
!
```

```
! 3) System-wide environment variables defined in the file
!    /usr/lib/GKS/.GKSdefaults
!
!
! The allowed syntax in this file is :
! ---------------------------------
!
! Comments start with  ! or  #  character
!       Space, tabs, and  "  characters are ignored
!       Associations are done by the  =  (equal) or  :  (colon) character
!
```

3.7 Environment Variable Types

The DEC GKS environment variables can be divided into two groups:

- General DEC GKS environment variables
- Graphics-handler environment variables

The following section describes the general DEC GKS environment variables. For information on the graphics-handler environment variables, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

3.7.1 General Environment Variables

Table 3–1 lists the general environment variables available with DEC GKS.

Table 3–1 General Environment Variables for DEC GKS

Variable	Value	Description
GKSasf	INDIVIDUAL or BUNDLED	Specifies the default aspect source flag (ASF) setting to be either BUNDLED or INDIVIDUAL. The predefined value is INDIVIDUAL.
GKSconid	String containing any valid workstation connection identifier	Specifies the default workstation connection identifier to be used in the call to OPEN WORKSTATION, if the caller passes conid 0. The predefined setting is platform-dependent.
GKSdefmode	ASAP, BNIG, BNIL or ASTI	Specifies the default deferral mode to be ASAP (as soon as possible), BNIG (before the next interaction globally), BNIL (before the next interaction locally), or ASTI (at some time). The predefined value is defined in the workstation description table.
GKSerrfile	String containing any valid error file specification	Specifies the default error file to be used in the OPEN GKS function, if the user passes a NIL (0) error file descriptor. On ULTRIX systems, this is set to stderr by default.
GKSerror	ON or OFF	Specifies whether the default standard error checking is ON or OFF. The predefined value is ON. Note that if you turn error checking OFF, you may improve overall DEC GKS throughput, but DEC GKS may terminate in an uncontrolled way.
GKSirg	SUPPRESSED or ALLOWED	Specifies the default implicit regeneration mode (IRG) to be set to either SUPPRESSED or ALLOWED. The predefined value is defined in the workstation description table.

(continued on next page)

Table 3–1 (Cont.) General Environment Variables for DEC GKS

Variable	Value	Description
GKSmetafile_type	GKSM or GKS3	Specifies the dimension of the metafile output. The value GKSM is for two-dimensional metafiles; GKS3 is for three-dimensional metafiles. The default value is GKS3.
GKSndc_clip	ON or OFF	Specifies the default NDC clipping to ON or OFF. The predefined value is ON.
GKSstroke_font1	String containing the file path for stroke font1	Specifies the default stroke font 1 to be used.
GKSwstype	String containing any valid workstation type number	Specifies the default workstation type to be used in the call to OPEN WORKSTATION, if the caller passes wstype 0. The predefined setting is platform-dependent.

3.8 Error Handling

The following sections contain information on error codes and error files.

3.8.1 Error Codes

Each DEC GKS function call returns a DEC GKS error value to the calling routine. All the returned error values are defined in the language file /usr/include/GKS/gks.f.

The function call can include a check of the returned status. If the function has executed successfully, the value NO_ERROR (0) is returned. If this value is not returned, the function has failed to execute successfully. See Appendix A for information about the DEC GKS errors returned. For the FORTRAN language, the error value is only accessible with a nonstandard extension of the DEC FORTRAN binding.

3.8.2 Error Files

Error messages are normally written to an error logging file. The application can specify the file name to the OPEN GKS function by using a logical unit number (LUN). The LUN can be used in any of the following ways:

- Explicitly: Use a FORTRAN OPEN function to associate a LUN with a device. Pass the LUN to the OPEN GKS call.
- Implicitly: Pass a LUN in the range 1 to 255, without explicitly opening an error file. DEC GKS writes error messages to a file named fort.*n*, where *n* is the 3-digit representation of the LUN.
- Default: Pass a LUN of 0. This opens the default error file. DEC GKS attempts to translate the environment variable GKSerrfile. If no translation exists, DEC GKS uses stderr.

If no messages have been written to the error file, a call to CLOSE GKS deletes the file.

3.9 Configuration Files

A configuration file contains the list of workstations to be linked with a DEC GKS application. There are two configuration files:

- gksconfig.c—for all device handlers except the DECwindows XUI handler
- gks_decw_config.c—for all device handlers except the Motif handler

You can use either of the two files, but you cannot use both. If you use the default configuration file (gksconfig.c) included in the DEC GKS libraries, all the device handlers supplied by Digital (except DECwindows XUI) will be linked into the program. If you use the gks_decw_config.c configuration file, all the device handlers supplied by Digital (except Motif) will be linked into the program. These files allow you to use numerous device handlers without relinking your program. However, this usually results in longer link times and larger executable images than are necessary. To reduce link time and image size, you can customize these files at either the system or user level. In either case, you customize the configuration file by changing either the INCLUDE macro to EXCLUDE, or vice versa for each device handler specified in the file. For a list of the workstation handlers and more information on the INCLUDE and EXCLUDE macros, see the configuration file gksconfig.c or gks_decw_config.c.

3.9.1 Customizing the Configuration File at System Level

To customize the file at the system level, edit the configuration file and exclude those handlers you do not wish to have included automatically in any program. Compile the file to create the new configuration module and use the command `ar(1)` to replace the file in the library /usr/lib/libGKS.a. When you replace the configuration module, other users must create their own copies of the configuration file (and link to it) to include handlers not contained in the system version of the file.

3.9.2 Customizing the Configuration File at User Level

To customize the file at the user level, make a private copy of the configuration file and edit it to include only the desired handlers. Compile the private copy of the file to create the new configuration module and link this private module before linking to the DEC GKS libraries.

Control Functions

Insert tabbed divider here. Then discard this sheet.

4
Control Functions

The control functions establish the DEC GKS and workstation environments, and control the workstation surface.

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 usually start a GKS application by calling the functions OPEN GKS, OPEN WORKSTATION, and ACTIVATE WORKSTATION. These functions initiate 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 (such as input requests and output generation). The following sections describe 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 describes 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, output, or 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 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 Motif®, PostScript®, CGM, and DDIF™ (Digital Document Interchange Format). If you are uncertain which devices your DEC GKS programs will use, you should review the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*. 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 GKS state list.

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

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 on.

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 capabilities, output capabilities, or both. (The GMO, GMI, GWISS 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. Table 4–1 describes the six workstation categories.

Table 4–1 Workstation Categories

Category	Description
GOUTPT	A workstation of the category GOUTPT can only display graphic images on a single display surface. A workstation of this category can process all output functions. Because the generalized drawing primitive (GDP) functions are device-dependent, not all GDPs can be displayed on all output workstations. For more information concerning GDPs, see Chapter 5. For a list of the supported GDPs for a particular output device, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.
GINPUT	A workstation of the category GINPUT can only accept input, which must be accepted by at least one type of logical input device. A workstation of this category cannot accept the generation of graphic images by DEC GKS output functions. For more information concerning input functions, see Chapter 9.
GOUTIN	A workstation of the category GOUTIN combines the capabilities of GOUTPT and GINPUT 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, see Chapter 9.

(continued on next page)

Table 4–1 (Cont.) Workstation Categories

Category	Description
GMO	A workstation of the category GMO (Metafile Output) stores image-specific data in a file for use in reproducing the graphic image at a later time, perhaps in another application program. For more information concerning metafiles, see Chapter 10.
GMI	A workstation of the category GMI (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 GMO workstation. For more information concerning metafiles, see Chapter 10.
GWISS	A workstation of the category GWISS (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, see Chapter 8.

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, use several devices, or 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:

- GKCL—GKS is closed.
- GKOP—GKS is open.
- WSOP—At least one workstation is open.
- WSAC—At least one workstation is active.
- SGOP—A segment is open.

The following sections describe the DEC GKS operating states at various points in a program.

Open GKS

Before you invoke DEC GKS, the operating state value is GKCL. When DEC GKS is closed, you can call INQUIRE OPERATING STATE VALUE, which returns the current operating state; you can call 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 highlighted in this paragraph), the call generates an error message. For more information on inquiry functions, see Chapter 11; for more information on error codes, see Appendix A.

To perform more tasks using DEC GKS, you must set the operating state to GKOP. To do this, call to the control function OPEN GKS, and pass to the function the name of an error log file so DEC GKS knows where to write error messages. If you specify the default error file (or the value 0), and have not redefined that environment option, 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 OPEN GKS, you have also allowed access to the GKS **state list**. The GKS state list contains entries that designate changeable information reflecting the current status of DEC GKS (such as the set of open workstations, the current normalization number, and the current character height.)

Once DEC GKS is open, you can then specify output attributes (see Chapter 6, Attribute Functions), set normalization transformations (see Chapter 7, Transformation Functions), obtain values from the GKS state list, and obtain values from the DEC GKS and workstation description tables (see Chapter 11). If you attempt to call other functions, DEC GKS generates an error message.

Open a Workstation

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 GKOP to WSOP (at least one workstation open). To accomplish this, call OPEN WORKSTATION and pass a numeric **workstation identifier**, a physical device name or connection identifier, and a workstation type. (See OPEN WORKSTATION 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 as defined by DEC GKS, whether the picture on the surface represents all the requests for output made thus far by the application program, and so on. 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 workstation. If you attempt to call these functions, DEC GKS produces an appropriate error message.

Activate a Workstation

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 WSOP to WSAC (at least one workstation active). To activate a workstation, call the control function ACTIVATE WORKSTATION, 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 OPEN GKS, CLOSE GKS, or CLOSE SEGMENT. If you attempt to call these functions, DEC GKS produces an error message.

Create a Segment

When you create a segment using the function CREATE SEGMENT, the DEC GKS operating state changes from WSAC to SGOP (segment open). You must pass a segment name to the CREATE SEGMENT function. 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 copy segments only 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 information about the current state of the segment, such as the segment name, the set of associated workstations, and the detectability of the segment.

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

Close a Segment

When you close the open segment using the CLOSE SEGMENT function, the DEC GKS kernel changes the operating state from SGOP to WSAC.

Deactivate and Close a Workstation

Calling the function DEACTIVATE WORKSTATION deactivates the specified workstation. If you deactivate the last active workstation, the kernel changes the DEC GKS operating state from WSAC to WSOP. Similarly, if you close the last open workstation (using the function CLOSE WORKSTATION), the kernel changes the DEC GKS operating state to GKOP.

Close GKS

The final call in a single DEC GKS session should be to CLOSE GKS; after the call, access to the DEC GKS environment is closed and your DEC GKS session ends.

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 correct order. If you do not, your DEC GKS session does not end properly.

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.

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 the changes previously requested by the application program. DEC GKS allows a workstation to delay the actions requested by a program 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 to improve overall rate of transmission, if a given workstation supports such buffering. Other times, you can release buffered output so 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:

- ASAP—Generates output as soon as possible.
- BNIG—Generates output before the next interaction globally.
- BNIL—Generates output before the next interaction locally.
- ASTI—Generates output at some time (as defined by workstation).

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

Depending on the capabilities of the workstation, it can defer output at any level up to the level specified in the call to SET DEFERRAL 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 ASAP in a call to SET DEFERRAL STATE, the workstation must generate output as soon as possible. If you specify BNIG, the workstation can defer output at either ASAP or BNIG, depending on its capabilities. If you specify BNIL, the workstation can defer output on any level up to and including BNIL, depending on its capabilities. If you specify 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 SET DEFERRAL STATE. To determine the default deferral state of a given workstation type, you can call INQUIRE DEFAULT DEFERRAL STATE VALUES. To determine the current state of the deferral mode, you can call INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES.

Writing applications with other graphics programs, you need to “flush the output buffer” 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 INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES. To release deferred output without updating the screen in any other way, call the function UPDATE WORKSTATION and pass the argument GPOSTP. For example, DECwindows, Motif, and ReGIS devices such as the VT240™, VT330™, and VT340™ 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 segment attribute, or the current workstation window or viewport, the workstation can either make the change to the surface dynamically (IMM) or can implicitly regenerate the entire picture to comply with the requested change (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 INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES or INQUIRE DYNAMIC MODIFICATION OF WORKSTATION ATTRIBUTES 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, only the output primitives in the picture affected by the change are regenerated and the surface does not become out of date. For example, for many of the supported workstations, a call to the function SET COLOUR REPRESENTATION (see Chapter 6, 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 example, for many of the supported workstations, a call to the function SET POLYLINE REPRESENTATION (see Chapter 6, 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 INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES to determine if the workstation is allowing regenerations (GALLOW) or suppressing them (GSUPPD).

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 control function SET DEFERRAL 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 UPDATE WORKSTATION, passing GPERFO, to perform the single implicit regeneration. Remember that if you call UPDATE WORKSTATION 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 GEMPTY. Under certain conditions, when you are working with different clipping rectangles and generalized drawing primitives (GDPs), the entry may contain GNEMPT when the surface is actually empty. In such situations, when the entry contains GNEMPT, 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 UPDATE WORKSTATION (passing GPERFO as an argument). If the new frame entry is GNO, you can update the surface without the fear of losing primitives not contained in segments. If the new frame entry is GYES, a call to UPDATE WORKSTATION with the GPERFO argument will cause an implicit regeneration, causing all primitives not contained in segments to be lost.

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:

INQUIRE DYNAMIC MODIFICATION OF WORKSTATION ATTRIBUTES
INQUIRE LEVEL OF GKS
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES
INQUIRE OPERATING STATE VALUE
INQUIRE SET OF ACTIVE WORKSTATIONS
INQUIRE SET OF OPEN WORKSTATIONS
INQUIRE WORKSTATION CATEGORY
INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES
INQUIRE WORKSTATION MAXIMUM NUMBERS
INQUIRE WORKSTATION STATE
INQUIRE WORKSTATION CONNECTION AND TYPE

For more information concerning device-independent programming, see the *DEC GKS User's Guide*.

4.4 Function Descriptions

This section describes the DEC GKS control functions in detail.

ACTIVATE WORKSTATION

Operating States

WSOP, WSAC

Syntax

GACWK (WKID)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier

Description

The ACTIVATE WORKSTATION function activates the specified workstation, allowing all subsequently generated output to be sent to the workstation. You must open DEC GKS and the workstation you wish to activate before calling this function. If the newly activated workstation is the only active workstation, DEC GKS changes the operating state from WSOP (at least one workstation open) to WSAC (at least one workstation active).

See Also

Example 4–1 for a program example using the ACTIVATE WORKSTATION function

CLEAR WORKSTATION

Operating States

WSOP, WSAC

Syntax

GCLRWK (WKID, COFL)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
COFL	Integer (constant)	Read	Control flag indicating condition under which DEC GKS clears the surface

Constants

Defined Argument	Constant	Description
COFL	GCONDI	Clear if the surface is not empty.
	GALWAY	Clear the workstation always.

Description

The CLEAR WORKSTATION function generates all deferred output and clears the display surface.

This function performs the following tasks:

1. Generates all deferred output (see the SET DEFERRAL STATE function).
2. If the *display surface* entry in the workstation state list is NOT EMPTY, this function always clears the surface. If the *display surface* entry is EMPTY, this function only clears the surface if you specify CLEAR ALWAYS as an argument. If no other workstations are associated with the segment, the segment is deleted.

After executing this function, DEC GKS sets the *display surface* entry in the workstation state list to EMPTY, the *workstation transformation update* entry to NOT PENDING, and the *new frame necessary at update* entry to NOT NECESSARY.

See Also

Example 4–1 for a program example using the CLEAR WORKSTATION function

CLOSE GKS

Operating States

GKOP

Syntax

GCLKS ( )

Description

The CLOSE GKS function releases the DEC GKS buffers, closes the error log file, and deletes the file if it is empty. The function also releases the DEC GKS description table, the GKS state list, and the workstation description tables. You must end each DEC GKS session with a call to this function.

You must call both the DEACTIVATE WORKSTATION function for each active workstation and the CLOSE WORKSTATION function for each open workstation before you call CLOSE GKS. If you do not, DEC GKS logs an error message.

A call to this function changes the DEC GKS operating state from GKOP (GKS open) to GKCL (GKS closed).

See Also

DEACTIVATE WORKSTATION
OPEN GKS
Example 4–1 for a program example using the CLOSE GKS function

CLOSE WORKSTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GCLWK (WKID)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier

Description

The CLOSE WORKSTATION function updates the workstation (equivalent to a call to the UPDATE WORKSTATION function with the regeneration mode argument), closes a workstation opened by a previous call to the OPEN WORKSTATION function, and releases the specified workstation's state list.

This function deassigns the channel used for both input and output to the device and removes the workstation from the set of open workstations in the GKS state list.

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

Be sure to deactivate a workstation with a call to the DEACTIVATE WORKSTATION function before you attempt to close a workstation with this function. If you do not, DEC GKS logs an error message.

See Also

DEACTIVATE WORKSTATION
OPEN WORKSTATION
Example 4–1 for a program example using the CLOSE WORKSTATION function

DEACTIVATE WORKSTATION

Operating States

WSAC

Syntax

GDAWK (WKID)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier

Description

The DEACTIVATE WORKSTATION function deactivates a specific workstation so subsequent output will not be sent to that workstation. This function removes the workstation from the set of active workstations in the 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 WSAC (at least one workstation active) to 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 error message.

See Also

ACTIVATE WORKSTATION
Example 4–1 for a program example using the DEACTIVATE WORKSTATION function

ESCAPE

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GESC (FCTID, LIDR, IDR, MLODR, LODR, ODR)

Argument	Data Type	Access	Description
FCTID	Integer (constant)	Read	Escape function identifier.
LIDR	Integer	Read	Number of elements in the input data record. This is typically set to the *LDR* argument value returned by the PACK DATA RECORD function.
IDR(LIDR)	Character*80	Read	Input data record.
MLODR	Integer	Read	Number of elements in output data record.
LODR	Integer	Write	Number of elements written to output data record. This is typically set to the *LDR* argument value usedby the UNPACK DATA RECORD function.
ODR(MLODR)	Character*80	Write	Output data record.

Constants

Escape Identifier	Description
GESP	Set Display Speed
GEP	Generate Hardcopy of Workstation Surface
GEB	Beep
GEPOPW	Pop Workstation
GEPSHW	Push Workstation
GESEHM	Set Error Handling Mode
GESVE	Set Viewport Event
GEAWC	Associated Workstation Type and Connection ID
GESCL	Software Clipping
GESWM	Set Writing Mode
GESLC	Set Line Cap Style

Escape Identifier	Description
GESLJ	Set Line Join Style
GESEC	Set Edge Control Flag
GESET	Set Edge Type
GESEW	Set Edge Width Scale Factor
GESECI	Set Edge Color Index
GESEI	Set Edge Index
GESEA	Set Edge Aspect Source Flag
GEBTB	Begin Transformation Block
GEETB	End Transformation Block
GESSHM	Set Segment Highlighting Method
GESHM	Set Highlighting Method
GBTB3	Begin Transformation Block 3
GESER	Set Edge Representation
GESWT	Set Window Title
GESRS	Set Reset String
GESCS	Set Cancel String
GESES	Set Enter String
GESIB	Set Icon Bit Maps
GEIWM	Inquire Current Writing Mode
GEILC	Inquire Current Line Cap Style
GEILJ	Inquire Current Line Join Style
GEIEA	Inquire Current Edge Attributes
GEIVD	Inquire Viewport Data
GEIS	Inquire Current Display Speed
GEILEI	Inquire List of Edge Indexes
GEISE	Inquire Segment Extent
GEIWID	Inquire Window Identifiers
GEISHM	Inquire Segment Highlighting Method
GEIHM	Inquire Highlighting Method
GEIPBI	Inquire Pasteboard Identifier
GEIMBI	Inquire Menu Bar Identifier
GEISHI	Inquire Shell Identifier
GEILE	Inquire List of Available Escapes
GEIDS	Inquire Default Display Speed
GEILCJ	Inquire Line Cap and Join Facilities
GEIEF	Inquire Edge Facilities
GEIPER	Inquire Predefined Edge Representation
GEIMEB	Inquire Maximum Number of Edge Bundles
GEILH	Inquire List of Highlighting Methods
GIER	Inquire Edge Representation

Escape Identifier	Description
GEMNW	Evaluate NDC Mapping of a WC Point
GEMDN	Evaluate DC Mapping of an NDC Point
GEMWN	Evaluate WC Mapping of an NDC Point
GEMND	Evaluate NDC Mapping of a DC Point
GEIGEX	Inquire Extent of a GDP
GECON	Set Connection Identifier String
GESDB	Set Double Buffering
GESBPM	Set Background Pixmap
GEIDBM	Inquire Double Buffer Pixmap
GEIBGM	Inquire Background Pixmap

Description

The ESCAPE function invokes a specified escape function. This function provides a method for DEC GKS to access capabilities of a specific workstation that are not fully utilized by other functions.

For example, the DEC GKS implementation uses the ESCAPE function to produce a hardcopy dump of a VT125 or VT240 terminal screen, or to set the LVP16 plotter pen speed.

The *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* describes the support for the DEC GKS escape functions. For more information concerning the available escapes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

The typical sequence of calls to use with the ESCAPE function is as follows:

1. Call the PACK DATA RECORD function to create the input data record.
2. Call the ESCAPE function.
3. Call the UNPACK DATA RECORD function to unpack the output data record returned by the ESCAPE function.

When calling the ESCAPE function, the application program needs to pass an input data record and an output data record. FORTRAN data records that are input arguments must be packed before you call the function, and FORTRAN data records that are output arguments must be unpacked after the function returns the information. Use the PACK DATA RECORD function to pack the data record, and use the UNPACK DATA RECORD function to unpack the data record.

When data records are used as arguments to functions, there are also two associated arguments that describe the data record size. One size argument is the dimension of the data record as specified in the FORTRAN declaration of the data record. The second size argument specifies how many elements are contained in the data record.

FORTRAN data records must be declared as follows:

```
CHARACTER*80 IDATAR(N)
```

In this declaration, *N* specifies the dimension of the data record. For example, to declare a data record of dimension 100, use the following syntax:

```
CHARACTER*80 IDATAR(100)
```

One way to describe *IDATAR* is an array of 100 elements; each element is of data type CHARACTER*80.

To construct the data records required by the FORTRAN escape function, you must first look up the specific escape in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for binding-independent information about the escape. You then need to use this information in terms of FORTRAN data records. For example, you should think of the *in_data* and *out_buffer* components listed in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* as arrays of integers, reals, and characters.

4.4.1 FORTRAN Escapes

The following code segments demonstrate how to convert the binding-independent information in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* to FORTRAN-specific information.

The example describes how the Inquire Segment Extent escape would be used by an application using the FORTRAN binding. You should review the description of this escape in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*, as well as the ESCAPE, PACK DATA RECORD, and UNPACK DATA RECORD functions described in this manual.

The Inquire Segment Extent escape requires an input data record as well as an output data record. The input data record is named *IDATAR* and the output data record is named *ODATAR*.

The first step is to declare these two data records. The *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* states that the *in_data* argument has four components, and from the description you can determine that the relevent information for FORTRAN is two integers. The component description is interpreted as follows:

2 integers
0 reals
0 character strings

Using this information, declare *IDATAR* to have dimension 2, by using the following formula:

```
2 integers + 0 reals + 0 character strings = 2
```

Additionally, the application program will need an integer array of dimension 2. The data declaration is as follows:

```
CHARACTER*80 IDATAR(2)
INTEGER IA(2)
```

The application program will also need the following variables, which will be used in the call to PACK DATA RECORD:

```
INTEGER IWKID, ISEGID, IL, RL, SL, MLDR, ERRIND, LDR, LSTR(1)
REAL RA(1)
CHARACTER*80 STR(1)
```

Construct the input data record by placing the workstation identifier and the segment identifier in an integer array and then calling the PACK DATA RECORD function. Note that you must specify the dimension of the input data record, as well as counts for the integers, reals, and strings as arguments to the PACK DATA RECORD function. The data declaration is as follows:

```
IL = 2
IA(1) = IWKID
IA(2) = ISEGID
RL = 0
SL = 0
MLDR = 2
CALL GPREC (IL,IA,RL,RA,SL,LSTR,STR,MLDR,ERRIND,LDR,IDATAR)
```

The PACK DATA RECORD function returns the packed data record, *IDATAR*, and the number of elements contained in the data record (in the argument *LDR*).

You must then determine the dimension of the data record returned by the ESCAPE function. The function returns a packed data record, so you must determine the sizes of the arrays needed for unpacking the data record. Again, look at the description of the *out_data* argument in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*. It can be interpreted as follows:

1 integer
4 reals
0 character strings

Therefore, the dimension of *ODATAR* is:

```
1 integer + 4 reals + 0 character strings = 5
```

You need an integer array of dimension 1 and a real array of dimension 4. The declarations are as follows:

```
CHARACTER*80 ODATAR(5)
INTEGER IAOUT(1)
REAL RAOUT(4)
```

Additonally, the application program needs the following variables:

```
INTEGER DODR, LODR, IIL, IRL, ISL, IERR
REAL XMIN, XMAX, YMIN, YMAX
```

Specify the escape function identifier and the dimension of the output data record, and call the escape:

```
FCTID = -303
DODR = 5
CALL GESC (FCTID,LDR,IDATAR,DODR,LODR,ODATAR)
```

You must specify the sizes of the arrays and call the UNPACK DATA RECORD function to get the data returned by the Inquire Segment Extent escape. Note that you pass the size argument (*LODR*) returned by the ESCAPE function to the UNPACK DATA RECORD function.

```
IIL = 1
IRL = 4
ISL = 0
CALL GUREC (LODR,ODATAR,IIL,IRL,ISL,ERRIND,IL,IAOUT,RL,RAOUT,SL,LSTR,
            STR)
```

The application can now use the segment extent data.

```
IERR = IAOUT(1)
XMIN = RAOUT(1)
XMAX = RAOUT(2)
YMIN = RAOUT(3)
YMAX = RAOUT(4)
```

The application should always check the error status arguments *ERRIND* and *IERR* to be sure the program executed successfully.

See Also

OPEN WORKSTATION for the Set DEC GKS Connection Identifier String escape
PACK DATA RECORD
UNPACK DATA RECORD
Example 4–2 for a program example using the ESCAPE function
Example 4–4 for a program example using the Set Connection Identifier String escape

MESSAGE

Operating States

WSOP, WSAC, SGOP

Syntax

GMSG (WKID, MESS)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
MESS	Character*(*)	Read	Text of message

Description

The MESSAGE function 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. This function may have a local effect on the workstation. For example, the message might request that the operator change the paper in a plotter before a picture is generated.

See the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for more information on workstation-specific capabilities.

See Also

Example 9–1 for a program example using the MESSAGE function

MESSAGE (FORTRAN–77 Subset)

Operating States

WSOP, WSAC, SGOP

Syntax

GMSGS (WKID, LSTR, MESS)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
LSTR	Integer	Read	Number of characters in the string
MESS	Character*80	Read	Text of message to be delivered to the specified workstation

Description

The MESSAGE function 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. This function may have a local effect on the workstation. For example, the message might request that the operator change the paper in a plotter before a picture is generated.

See the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for more information on workstation-specific capabilities.

See Also

Example 9–1 for a program example using the MESSAGE function

OPEN GKS

Operating States

GKCL

Syntax

GOPKS (ERRFIL, BUFA)

Argument	Data Type	Access	Description
ERRFIL	Integer	Read	The logical unit number (LUN) of a device or file that points to the error log file. If the value 0 is passed, DEC GKS uses a default error file. DEC GKS first attempts to translate the VMS logical name GKS$ERRFILE or the ULTRIX environment variable GKSerrfile. If no translation exists, DEC GKS uses the translation of the VMS logical name SYS$ERROR or the ULTRIX environment variable stderr.
BUFA	Integer	Read	Dummy argument for conformance with the GKS standard.

Description

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

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

When using DEC GKS, you usually call this function first. All functions except EMERGENCY CLOSE, ERROR HANDLING, ERROR LOGGING, and INQUIRE OPERATING STATE VALUE require at least the GKOP operating state.

See Also

CLOSE GKS
Example 4–1 for a program example using the OPEN GKS function

OPEN WORKSTATION

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GOPWK (WKID, CONID, WTYPE)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
CONID	Integer	Read	LUN of connection identifier. To specify a string as the connection identifier, use the Connection Identifier String escape (–440).
WTYPE	Integer	Read	Specifies workstation type.

Description

The OPEN WORKSTATION function initializes a workstation for use by DEC GKS, permitting subsequent access to the specified workstation's state list.

This function associates the workstation identifier with a particular device of a specified type, and initializes the workstation. If establishing the first open workstation, this function changes the DEC GKS operating state from GKOP (GKS open) to WSOP (at least one workstation open).

This function clears the display surface of previously generated images. You must call this function, followed by a call to the ACTIVATE WORKSTATION function, before you attempt to generate output to this workstation.

For information on how to specify a string as the connection identifier, see the ESCAPE function.

The Set DEC GKS Connection Identifier String (–440) escape allows you to specify a string as the DEC GKS connection identifier. This escape is supported only by the FORTRAN binding.

–440 SET DEC GKS CONNECTION IDENTIFIER STRING

Operating States: GKOP, WSOP, WSAC, SGOP
Constant: GECON
Supporting Workstations: All workstations

Sets the connection identifier string to be used in the next OPEN WORKSTATION (GOPWK) call. The connection identifier string will be deleted by the next GOPWK, so a new string must be set before each GOPWK call.

ESCAPE Arguments:

Argument	Required Value
function_id	–440
in_data	Seven components: 0 0 1 0 0 (address of) length of new string (address of) conid string
in_data_size	28 bytes
out_buffer	null
record_buffer_length	NA
record_size	0 bytes

See Also

ACTIVATE WORKSTATION
Example 4–1 for a program example using the OPEN WORKSTATION function
Example 4–4 for a program example using the Set Connection Identifier String escape

REDRAW ALL SEGMENTS ON WORKSTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GRSGWK (WKID)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier

Description

The REDRAW ALL SEGMENTS ON WORKSTATION function clears the screen and redraws all defined, visible segments.

This function performs the following tasks:

1. Generates all deferred output (see the SET DEFERRAL STATE function).
2. If the *display surface empty* entry in the workstation state list is NOT EMPTY, 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 list entry to NOT PENDING, and the *new frame necessary at update* state list entry to NOT NECESSARY.

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 the UPDATE WORKSTATION function.

See Also

UPDATE WORKSTATION
Example 8–1 for a program example using the REDRAW ALL SEGMENTS ON WORKSTATION function

SET DEFERRAL STATE

Operating States

WSOP, WSAC, SGOP

Syntax

GSDS (WKID, DEFMOD, REGMOD)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEFMOD	Integer (constant)	Read	Maximum allowable deferral mode
REGMOD	Integer (constant)	Read	Implicit regeneration mode

Constants

Defined Argument	Constant	Description
DEFMOD	GASAP	Generate images as soon as possible.
	GBNIG	Generate images before the next interaction globally, or before a sample or event input occurs.
	GBNIL	Generate images before the next interaction locally, or before a sample or event input occurs.
	GASTI	Generate images at some time. The exact time is determined by the workstation.
REGMOD	GSUPPD	Image regeneration is suppressed.
	GALLOW	Image regeneration is allowed.

Description

The SET DEFERRAL STATE function sets the workstation state list entries *deferral mode* and *implicit regeneration mode*.

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 the SET DEFERRAL STATE function. 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. Using this function, you can allow a workstation to defer output, or you can either suppress or allow implicit regenerations.

For example, if you specify ASAP in a call to this function, the workstation must generate output as soon as possible. If you specify BNIG, the workstation can defer output at either ASAP or BNIG, depending on its capabilities. If you specify BNIL, the workstation can defer output on any level up to and including BNIL, depending on its capabilities. If you specify 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 described in this paragraph, see the deferral mode argument description.)

The implicit regeneration mode determines whether implicit regenerations are allowed or suppressed. If you allow implicit regenerations, and the workstation supports implicit regeneration for the specified change, any pending or subsequent surface change requiring regeneration (for example, 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 NEWFRAME NECESSARY).

By suppressing implicit regenerations, you can make all necessary changes without altering the workstation surface. When you have requested all changes, call the UPDATE WORKSTATION function to perform all the 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.

See Also

UPDATE WORKSTATION
Example 5–1 for a program example using the SET DEFERRAL STATE function

UPDATE WORKSTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GUWK (WKID, REGFL)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
REGFL	Integer (constant)	Read	Identifies update regeneration mode

Constants

Defined Argument	Constant	Description
REGFL	GPOSTP	Postpone regeneration of image.
	GPERFO	Perform regeneration of image.

Description

The UPDATE WORKSTATION function generates all deferred output for the specified workstation and can also redisplay all visible segments.

If the *new frame necessary at update* entry in the workstation state list is NEWFRAME NECESSARY, and if you specify the value GPERFO to this function, it performs the following tasks:

1. Clears the screen if the *display surface empty* entry in the workstation state list is NOT EMPTY.
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 these tasks, DEC GKS sets the *display surface empty* entry in the workstation state list to EMPTY or to NOT EMPTY according to the current state of the workstation surface, the *workstation transformation update state* entry to NOT PENDING, and the *new frame necessary at update* entry to NOT NECESSARY.

However, if at the call to this function the *new frame necessary at update* entry in the workstation state list is NOT NECESSARY, or if you specify the value GPOSTP as an argument to this function, it initiates only the transmission of any deferred output.

See Also

Example 4–1 for a program example using the UPDATE WORKSTATION function

4.5 Program Examples

Example 4–1 illustrates the use of several control functions.

Example 4–1 CLEAR WORKSTATION and the GKS Control Functions

```
C  This program writes a text string to the screen, and then clears
C  the screen using the function CLEAR WORKSTATION.
C
C  NOTE: To keep the example concise, no error checking is performed.

      IMPLICIT NONE
      INCLUDE  'gks.f'

      INTEGER  DEVNUM
      INTEGER  ICL
      REAL     LARGER
      REAL     START_X
      REAL     START_Y
      REAL     TIME
      INTEGER  WS_ID

C  Open the GKS environment. Specifying 6 and 0 for the two arguments
C  tells DEC GKS to use the default values (output to the terminal
C  surface and the default GKS error file).

      CALL GOPKS (6, 0)

C  Open the workstation environment.  When you call this function,
C  you assign the workstation a numeric identifier (in this example,
C  the number 1), a device name (in this example, DEC GKS translates
C  the connection identifier environment option to determine the device
C  name), and a workstation type (in this example, DEC GKS translates the
C  workstation type environment option to determine the workstation type).

      WS_ID = 1

      CALL GOPWK (WS_ID, GCONID, GWSDEF)

C  When activating a workstation using the function ACTIVATE
C  WORKSTATION, use the workstation identifier you specified as the
C  first argument in the call to OPEN WORKSTATION (in this example,
C  the number 1).

      CALL GACWK (WS_ID)

C  Using the default windows and viewports, the TEXT function writes
C  a character string to the screen starting at the WC (0.1, 0.5).

      LARGER = 0.03
      START_X = 0.1
      START_Y = 0.5

      CALL GSCHH (LARGER)
      CALL GTX (START_X, START_Y,
     *'CLEAR WORKSTATION should erase this')
```

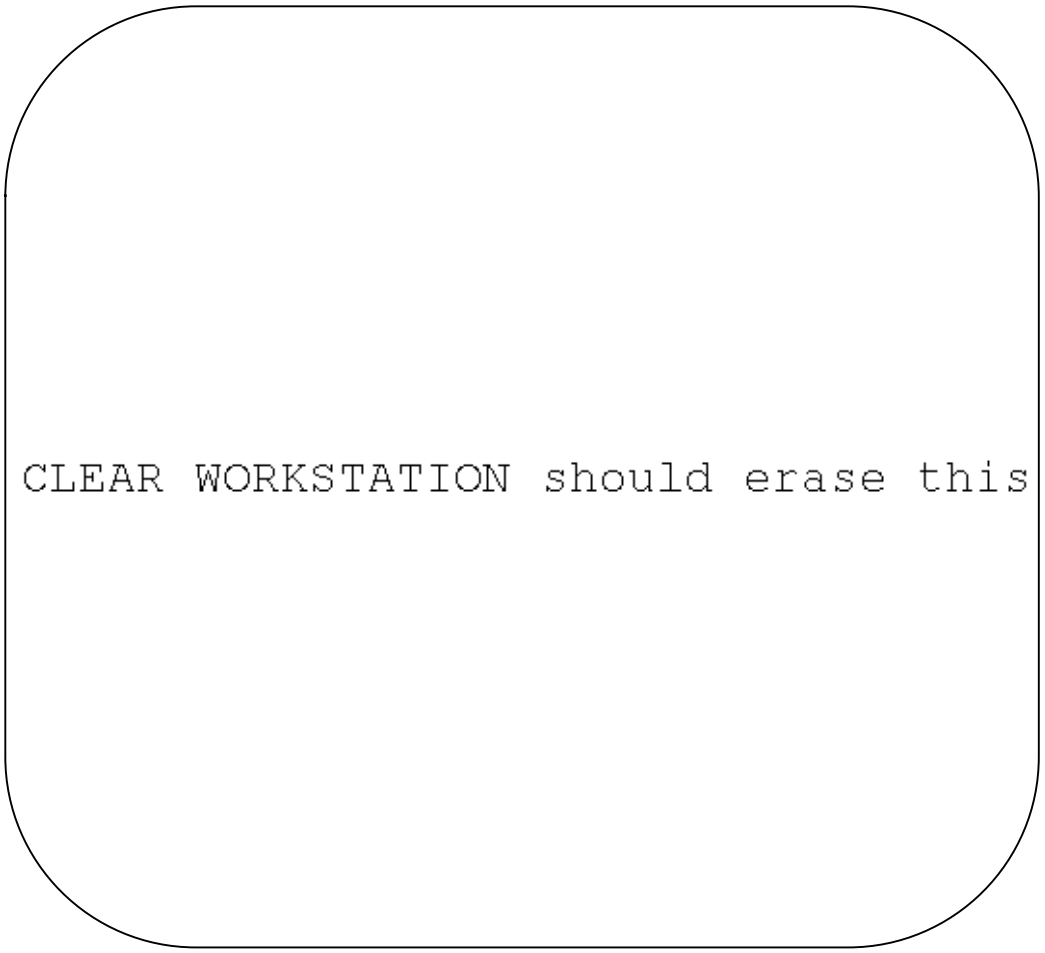
(continued on next page)

Example 4–1 (Cont.) CLEAR WORKSTATION and the GKS Control Functions

```
C     Release the deferred output.  Wait 10 seconds.

      CALL GUWK (WS_ID, GPOSTP)
      TIME = 10
      CALL GWAIT (TIME, WS_ID, ICL, DEVNUM)

C  The CLEAR WORKSTATION function, when passed the flag GCONDI,
C  clears the workstation under the condition that the surface
C  contains output primitives.  Since the previous function call
C  wrote a character string to the workstation surface, this call
C  clears the screen.

      CALL GCLRWK (WS_ID, GCONDI)

C  When deactivating and closing the open workstation, pass the
C  numeric workstation identifier previously specified in the call
C  to OPEN WORKSTATION (in this example, the value 1).

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 4–1 shows the first screen display on a VAXstation™ workstation running DECwindows software. When the program ends, the screen is cleared.

Figure 4–1 CLEAR WORKSTATION and the GKS Control Functions



ZK–4014A–GE

Example 4–2 illustrates the use of the ESCAPE function.

Example 4–2 Supported Escapes Program

```
C  This program opens GKS and the default workstation, inquiries the
C  workstation type, queries the list of escapes supported by the workstation,
C  tests a few of the escapes, and closes the workstation and GKS.

      PROGRAM ESCAPE

      IMPLICIT NONE
      INCLUDE 'gks.f'
```

(continued on next page)

Example 4–2 (Cont.) Supported Escapes Program

```
        INTEGER      CONID
        INTEGER      DOUBLE_BUFFER_FLAG
        INTEGER      ERRIND
        INTEGER      ESCAPE_LIST(100)
        INTEGER      K
        CHARACTER*30 TITLE /'DEC GKS puts life in perspective.'/
        INTEGER      TITLE_LEN /30/
        INTEGER      WS_ID /1/
        INTEGER      WSTYPE
        INTEGER      X_DISPLAY_ID
        INTEGER      X_WINDOW_ID

C  Open GKS and the default workstation.

        CALL GOPKS (0, 0)
        CALL GOPWK (WS_ID, 0, 0)

C  Inquire the workstation type.

        CALL GQWKC (WS_ID, ERRIND, CONID, WSTYPE)

C  Inquire the list of supported escapes.

        CALL INQUIRE_ESCAPES (WSTYPE, ESCAPE_LIST)
        IF (ESCAPE_LIST(1) .NE. 0) THEN
           CALL GECLKS ( )
           TYPE *, 'Error inquiring list of supported escapes ',
     1             ESCAPE_LIST(1)
           STOP
        END IF

C  Set the window title.

        CALL SET_WINDOW_TITLE (WS_ID, ESCAPE_LIST, TITLE, TITLE_LEN)

C  Turn double buffering on.

        DOUBLE_BUFFER_FLAG = 1

        CALL SET_DOUBLE_BUFFER (WS_ID, ESCAPE_LIST, DOUBLE_BUFFER_FLAG)

C  Inquire the X display identifier and X window identifier.

        CALL INQ_WINDOW_IDS (WS_ID, ESCAPE_LIST,
     1                       X_DISPLAY_ID, X_WINDOW_ID)

C  Close the workstation and GKS.

        CALL GCLWK (WS_ID)
        CALL GCLKS ( )

C  Print the workstation type with its supported escapes.

        WRITE (6,10) WSTYPE, ESCAPE_LIST(2), ESCAPE_LIST(3)
10      FORMAT (' ', '> Workstation type ', I3,
     1             ' supports ', I2, ' escapes (', I2, ' returned):')
        DO K = 3+1, 3+ESCAPE_LIST(3)
           WRITE (6,20) ESCAPE_LIST(K)
20         FORMAT (' ', '   ', I4)
        END DO

        END


C  This routine determines whether or not the specified escape identifier
C  (esc_id) is in the list of supported escapes (escape_list).
C  A value of TRUE is returned if the specified escape is supported.
C  A value of FALSE is returned if the specified escape is not supported.
```

(continued on next page)

Example 4–2 (Cont.) Supported Escapes Program

```
      LOGICAL FUNCTION ESC_SUPPORT (ESC_ID, ESCAPE_LIST)

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER   ESC_ID
      INTEGER   ESCAPE_LIST(*)
      INTEGER   N

      ESC_SUPPORT = .FALSE.
      DO N = 4, 3+ESCAPE_LIST(3)
         IF (ESC_ID .EQ. ESCAPE_LIST(N)) ESC_SUPPORT = .TRUE.
      END DO

      RETURN
      END

C  This routine inquires the list of supported escapes for the specified
C  workstation type.

      SUBROUTINE INQUIRE_ESCAPES (WSTYPE, IA)

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER      ERRIND
      INTEGER      IA (IIL)
      INTEGER      IIL
      PARAMETER (IIL = 100)
      INTEGER      IL
      CHARACTER*80 INDATAREC (100)
      INTEGER      IRL
      PARAMETER (IRL = 10)
      INTEGER      ISL
      PARAMETER (ISL = 10)
      INTEGER      LDR
      INTEGER      LSTR (ISL)
      INTEGER      LODR
      INTEGER      MLDR  /100/
      INTEGER      MLODR /100/
      CHARACTER*80 OUTDATAREC (100)
      REAL         RA (IRL)
      INTEGER      RL
      INTEGER      SL
      CHARACTER*80 STR (ISL)
      INTEGER      WSTYPE


C  Pack the escape data record.

      IA(1) = WSTYPE
      IL = 1
      RL = 0
      SL = 0

      CALL GPREC (IL,                  ! number of integers
     1            IA,                  ! integer array
     1            RL,RA,               ! number of reals
     1            SL,LSTR,STR,         ! number strings, string-length array
     1            MLDR,                ! number of  entries in datrec
     1            ERRIND,              ! error indicator
     1            LDR,                 ! number of datrec entries used
     1            INDATAREC)
```

(continued on next page)

Example 4–2 (Cont.) Supported Escapes Program

```
      IF (ERRIND .NE. 0) THEN
         TYPE *, 'Error packing escape data record, status = ', ERRIND
      END IF

C  Get a list of the escapes supported by the specified workstation.

      CALL GESC (-350,           ! escape function
     1           LDR,            ! number of elements in input datrec
     1           INDATAREC,      ! input data record
     1           MLODR,          ! number of elements in output datarec
     1           LODR,           ! number of elements written to datarec
     1           OUTDATAREC)     ! output data record

C  Unpack the list of supported escapes.

      CALL GUREC (LODR,          ! number of datrec entries used
     1            OUTDATAREC,    ! output data record
     1            IIL,           ! dimension of integer array
     1            IRL,           ! dimension of real array
     1            ISL,           ! dimension of string array
     1            ERRIND,        ! error indicator
     1            IL,            ! number of integers
     1            IA,            ! integer array
     1            RL,            ! number of reals
     1            RA,            ! real array
     1            SL,            ! number of strings
     1            LSTR,          ! string lengths array
     1            STR)           ! strings array

      IF (ERRIND .NE. 0) THEN
         TYPE *, 'Error unpacking escape data record, status = ',
     1           ERRIND
      END IF

      RETURN
      END

C  This routine sets double buffering ON or OFF as requested. Nothing
C  will be done if the escape is not supported by the workstation
C  type.

      SUBROUTINE SET_DOUBLE_BUFFER
     1           (WS_ID, ESCAPE_LIST, DOUBLE_BUFFER_FLAG)

      IMPLICIT NONE
      INCLUDE 'gks.f'
```

(continued on next page)

Example 4–2 (Cont.) Supported Escapes Program

```
      INTEGER      DOUBLE_BUFFER_FLAG
      INTEGER      ERRIND
      LOGICAL      ESC_SUPPORT
      INTEGER      ESCAPE_LIST(*)
      INTEGER      IA (IIL)
      INTEGER      IIL
      PARAMETER (IIL = 10)
      INTEGER      IL
      CHARACTER*80 INDATAREC (10)
      INTEGER      IRL
      PARAMETER (IRL = 10)
      INTEGER      ISL
      PARAMETER (ISL = 10)
      INTEGER      LDR
      INTEGER      LODR
      INTEGER      LSTR(ISL)
      INTEGER      MLDR  /10/
      INTEGER      MLODR /10/
      CHARACTER*80 OUTDATAREC (10)
      REAL         RA (IRL)
      INTEGER      RL
      INTEGER      SL
      CHARACTER*80 STR (ISL)
      INTEGER      WS_ID

C  Determine if the escape is supported by the workstation.

      IF (.NOT. (ESC_SUPPORT (-500, ESCAPE_LIST))) RETURN

C  Pack the escape data record.

      IA(1) = WS_ID
      IA(2) = DOUBLE_BUFFER_FLAG
      IL = 1
      RL = 0
      SL = 0

      CALL GPREC (IL,                   ! number of integers
     1            IA,                   ! integer array
     1            RL,RA,                ! number of reals
     1            SL,LSTR,STR,          ! number strings, string-length array
     1            MLDR,                 ! number of entries in datrec
     1            ERRIND,               ! error indicator
     1            LDR,                  ! number of datrec entries used
     1            INDATAREC)

      IF (ERRIND .NE. 0) THEN
         TYPE *, 'Error packing escape data record, status = ', ERRIND
      END IF

C  Set double buffering ON or OFF as requested.

      CALL GESC (-500,           ! escape function
     1           LDR,            ! number of elements in input datrec
     1           INDATAREC,      ! input data record
     1           MLODR,          ! number of elements in output datarec
     1           LODR,           ! number of elements written to datarec
     1           OUTDATAREC)     ! output data record

      RETURN
      END
```

(continued on next page)

Example 4–2 (Cont.) Supported Escapes Program

```
C  This routine inquires the X window and display identifiers of the GKS
C  workstation.  Nothing will be done if the escape is not supported by
C  the workstation type.

      SUBROUTINE INQ_WINDOW_IDS
     1            (WS_ID, ESCAPE_LIST, X_DISPLAY_ID, X_WINDOW_ID)

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER      ERRIND
      LOGICAL      ESC_SUPPORT
      INTEGER      ESCAPE_LIST(*)
      INTEGER      IA (IIL)
      INTEGER      IIL
      PARAMETER (IIL = 10)
      INTEGER      IL
      CHARACTER*80 INDATAREC (10)
      INTEGER      IRL
      PARAMETER (IRL = 10)
      INTEGER      ISL
      PARAMETER (ISL = 10)
      INTEGER      LDR
      INTEGER      LODR
      INTEGER      LSTR(ISL)
      INTEGER      MLDR  /10/
      INTEGER      MLODR /10/
      CHARACTER*80 OUTDATAREC (10)
      REAL         RA (IRL)
      INTEGER      RL
      INTEGER      SL
      CHARACTER*80 STR (ISL)
      INTEGER      WS_ID
      INTEGER      X_DISPLAY_ID
      INTEGER      X_WINDOW_ID


C  Determine if the escape is supported by the workstation.

      IF (.NOT. (ESC_SUPPORT (-304, ESCAPE_LIST))) RETURN

C  Pack the escape data record.

      IA(1) = WS_ID
      IL = 1
      RL = 0
      SL = 0

      CALL GPREC (IL,                  ! number of integers
     1            IA,                  ! integer array
     1            RL,RA,               ! number of reals
     1            SL,LSTR,STR,         ! number strings, string-length array
     1            MLDR,                ! number entries in datrec
     1            ERRIND,              ! error indicator
     1            LDR,                 ! number of datrec entries used
     1            INDATAREC)

      IF (ERRIND .NE. 0) THEN
         TYPE *, 'Error packing escape data record, status = ', ERRIND
      END IF
```

(continued on next page)

Example 4–2 (Cont.) Supported Escapes Program

```
C  Get the window identifiers.

        CALL GESC (-304,           ! escape function
     1             LDR,            ! number of elements in input datrec
     1             INDATAREC,      ! input data record
     1             MLODR,          ! number of elements in output datarec
     1             LODR,           ! number elements written to datarec
     1             OUTDATAREC)     ! output data record

C  Unpack the window identifiers.

        CALL GUREC (LODR,          ! number of datrec entries used
     1              OUTDATAREC,    ! output data record
     1              IIL,           ! dimension of integer array
     1              IRL,           ! dimension of real array
     1              ISL,           ! dimension of string array
     1              ERRIND,        ! error indicator
     1              IL,            ! number of integers
     1              IA,            ! integer array
     1              RL,            ! number of reals
     1              RA,            ! real array
     1              SL,            ! number of strings
     1              LSTR,          ! string lengths array
     1              STR)           ! strings array

        IF (ERRIND .NE. 0) THEN
           TYPE *, 'Error unpacking escape data record, status = ',
     1             ERRIND
        END IF

        X_DISPLAY_ID = IA(1)
        X_WINDOW_ID  = IA(2)

        TYPE *, 'X Display ID:', X_DISPLAY_ID
        TYPE *, 'X Window ID: ', X_WINDOW_ID

        RETURN
        END

C  This routine sets the window title to the specified string.  Nothing will
C  be done if the escape is not supported by the workstation type.

        SUBROUTINE SET_WINDOW_TITLE
     1             (WS_ID, ESCAPE_LIST, TITLE, TITLE_LEN)

        IMPLICIT NONE
        INCLUDE 'gks.f'

        INTEGER      ERRIND
        LOGICAL      ESC_SUPPORT
        INTEGER      ESCAPE_LIST(*)
```

(continued on next page)

Example 4–2 (Cont.) Supported Escapes Program

```
      INTEGER      IA (IIL)
      INTEGER      IIL
      PARAMETER (IIL = 10)
      INTEGER      IL
      CHARACTER*80 INDATAREC (10)
      INTEGER      IRL
      PARAMETER (IRL = 10)
      INTEGER      ISL
      PARAMETER (ISL = 10)
      INTEGER      LDR
      INTEGER      LODR
      INTEGER      LSTR (ISL)
      INTEGER      MLDR  /10/
      INTEGER      MLODR /10/
      CHARACTER*80 OUTDATAREC (10)
      INTEGER      RL
      REAL         RA (IRL)
      INTEGER      SL
      CHARACTER*80 STR (ISL)
      CHARACTER*(*) TITLE
      INTEGER      TITLE_LEN
      INTEGER      WS_ID

C  Determine if the escape is supported by the workstation.

      IF (.NOT. (ESC_SUPPORT (-202, ESCAPE_LIST))) RETURN

C  Pack the escape data record.

      IA(1) = WS_ID
      IL = 1
      RL = 0
      SL = 1
      LSTR(1) = TITLE_LEN
      STR(1)  = TITLE

      CALL GPREC (IL,                  ! number of integers
     1            IA,                  ! integer array
     1            RL,RA,               ! number of reals
     1            SL,LSTR,STR,         ! number strings, string-length array
     1            MLDR,                ! number of entries in datrec
     1            ERRIND,              ! error indicator
     1            LDR,                 ! number of datrec entries used
     1            INDATAREC)

      IF (ERRIND .NE. 0) THEN
         TYPE *, 'Error packing escape data record, status = ', ERRIND
      END IF

C  Set the window title to the specified string.

      CALL GESC (-202,           ! escape function
     1           LDR,            ! number of elements in input datrec
     1           INDATAREC,      ! input data record
     1           MLODR,          ! number of elements in output datarec
     1           LODR,           ! number of elements written to datarec
     1           OUTDATAREC)     ! output data record

      RETURN
      END
```

This program performs various escapes, depending on the workstation type. Example 4–3 shows the output from a VAXstation workstation running DECwindows software.

Example 4–3 VAXstation Output for Escape Program

```
Escape: Inquire Window Identifiers
  X Display ID:      1f8010
  X Window ID:       700013

Workstation type 211 supports 35 escapes (35 returned):
   -104
   -105
   -151
   -152
   -103
   -150
   -309
   -308
   -307
   -106
   -107
   -206
   -304
   -202
   -203
   -204
   -205
   -500
   -501
   -502
   -503
   -109
   -401
   -403
   -303
   -358
   -108
   -110
   -251
   -252
   -253
   -305
   -350
   -400
   -402
```

Example 4–4 illustrates the use of the Set Connection Identifier String escape.

Example 4–4 Using the FORTRAN Escape Function

```
      IMPLICIT NONE
      INCLUDE  'gks.f'

      INTEGER        ERRIND
      INTEGER        FCTID
      INTEGER        IA
      CHARACTER*80   IDR (10)
      INTEGER        IL
      INTEGER        KCONID
      INTEGER        LDR
      INTEGER        LIDR
      INTEGER        LODR
      INTEGER        LSTR (1)
      INTEGER        MLODR
      INTEGER        ODR
      INTEGER        RA
      INTEGER        RL
      INTEGER        SL
      CHARACTER *80  STR (1)
      INTEGER        WKID
      INTEGER        WSTYPE
      REAL           X (2)
      REAL           Y (2)
      REAL           Z (2)

C  Initialize GKS.

      CALL GOPKS  (1, 10000)
      CALL GSTXFP (1, 1)

C  Define the CONID string using a packed record and the FORTRAN binding
C  escape function -440.  This program uses a remote DECwindows server.

      IL = 0
      IA = 0
      RL = 0
      RA = 0
      SL = 1
      MLDR = 10
      LSTR(1) = LEN ('NODE::0.0')
      STR(1) = 'NODE::0.0'

      CALL GPREC (IL, IA, RL, RA, SL, LSTR, STR,
     *MLDR, ERRIND, LDR, IDR)

      FCTID = -440
      MLODR = 0
      LODR  = 0

      CALL GESC (FCTID, LDR, IDR, MLODR, LODR, ODR)

C  Open and activate the workstation.

      KCONID = 0
      WKID = 1
      WSTYPE = 211

      CALL GOPWK (WKID, KCONID, WSTYPE)
      CALL GACWK (WKID)
      CALL GSDS  (WKID, 0, 1)

C  Draw the object.

      CALL GSLWSC (2.0)
      CALL GSPLCI (2)
      CALL GCRSG (1)
```

(continued on next page)

Example 4–4 (Cont.) Using the FORTRAN Escape Function

```
      Z(1) =  0.0
      Z(2) =  0.0
      X(1) =  0.0
      Y(1) =  0.0
      X(2) =  1.0
      Y(2) =  1.0

      CALL GPL3 (2, X, Y, Z)
      CALL GCLSG
      PAUSE

20    CONTINUE
      CALL GDAWK (WKID)
      CALL GCLWK (WKID)
      CALL GCLKS
      END
```

Output Functions

Insert tabbed divider here. Then discard this sheet.

5
Output Functions

The DEC GKS output functions generate the basic components, or **primitives**, of all graphic pictures.

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 describe the topics in full detail.

5.1 Output and the DEC GKS Operating State

When 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 (see Section 4.1.2).

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

- Open DEC GKS (by calling OPEN GKS).
- Open at least one workstation (by calling OPEN WORKSTATION).
- Activate at least one workstation (by calling ACTIVATE WORKSTATION).

If you call an output function, DEC GKS generates the primitive on all active workstations. If you call an output function during the SGOP operating state, the output primitive becomes part of a segment. (For complete information concerning segments, see Chapter 8, Segment Functions.)

If you wish to output to an active workstation, the workstation must be of type OUTPUT, OUTIN, or MO (see Table 4–1). Only workstations of those categories support image generation. OUTPUT and OUTIN workstations generate output primitives on the workstation surface; MO workstations store information about the function call in a file. For more information concerning metafiles, see Chapter 10, Metafile Functions. For more information concerning workstation categories or the DEC GKS operating states, see Chapter 4, Control Functions.

5.2 Output Attributes

All the output primitives have **attributes** that are stored in the 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 example, depending on the current text attribute values, the positioning point passed to the output function 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, see Chapter 6, Attribute Functions.

5.3 Transformations and the DEC GKS Coordinate Systems

The DEC GKS transformation functions allow you to compose a picture, control how much of the picture is displayed on the workstation surface, orient the picture, and control how much of the workstation surface is used to display the picture.

When you request input and generate output on the workstation surface, you actually work with a number of coordinate systems. The image is transformed from one coordinate system to the next.

Using DEC GKS, you work with a geometric transformation pipeline. The pipeline consists of a number of transformations that affect various coordinate systems.

Note if you are working only with two-dimensional primitives, your WC system is an imaginary Cartesian coordinate system with the origin at (0, 0), and X and Y axes that extend to infinity in all directions. The two-dimensional WC plane is positioned at z = 0.

DEC GKS uses two separate transformations to translate your WC points to NDC points, and to translate your NDC points to device coordinate points. 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, see 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 ASAP (generates output as soon as possible), BNIG (generates output before the next interaction globally), BNIL (generates output before the next interaction locally), and ASTI (at some time).

You can specify a suggested level of deferral by calling the function SET DEFERRAL 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 SET DEFERRAL STATE.

For detailed information concerning SET DEFERRAL STATE and deferral, see 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:

INQUIRE GENERALIZED DRAWING PRIMITIVE
INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES
INQUIRE OPERATING STATE
INQUIRE PIXEL
INQUIRE PIXEL ARRAY
INQUIRE PIXEL ARRAY DIMENSIONS
INQUIRE SET OF ACTIVE WORKSTATIONS
INQUIRE TEXT EXTENT

For more information concerning device-independent programming, see the *DEC GKS User's Guide*.

5.6 Function Descriptions

This section describes the DEC GKS output functions in detail.

CELL ARRAY

Operating States

WSAC, SGOP

Syntax

GCA (PX, PY, QX, QY, DIMX, DIMY, ISC, ISR, DX, DY, COLIA)

Argument	Data Type	Access	Description
PX, PY	Real	Read	Cell array starting point in WC points (upper right corner)
QX, QY	Real	Read	Cell array ending point in WC points (lower left corner)
DIMX, DIMY	Integer	Read	Dimensions of color index array
ISC	Integer	Read	First column in color index array
ISR	Integer	Read	First row in color index array
DX	Integer	Read	Number of columns in color index array to be used
DY	Integer	Read	Number of rows in color index array to be used
COLIA (DIMX,DIMY)	Integer	Read	Color index array

Description

The CELL ARRAY function divides a designated rectangular area into cells, and displays each cell in a specified color or shade.

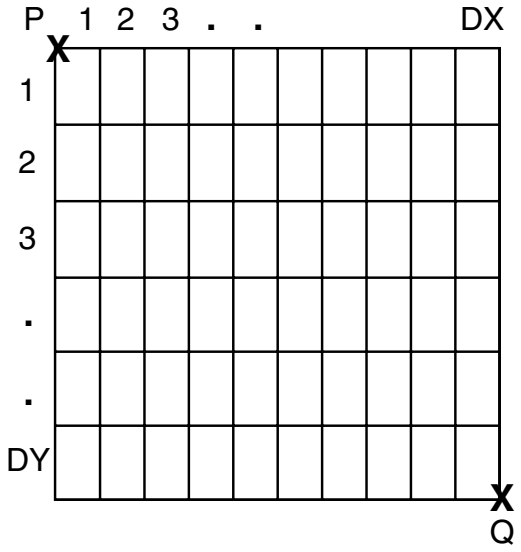
You pass a two-dimensional array containing color index values as one argument to this function. DEC GKS maps the color index values to corresponding cells within a rectangular area of the workstation surface. In addition to the color index array, you specify an offset into the color array (a starting element), and the number of array columns and 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 on the initial color index values for a given workstation, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

To alter the color associated with a certain index value, you can use the GKS function SET COLOUR REPRESENTATION.

The following figure illustrates how the cells are arranged in the cell array primitive.



See Also

SET COLOUR REPRESENTATION
Example 5–1 for a program example using the CELL ARRAY function

CELL ARRAY 3

Operating States

WSAC, SGOP

Syntax

GCA3 (CXA, CYA, CZA, DIMX, DIMY, ISC, ISR, DX, DY, COLIA)

Argument	Data Type	Access	Description
CXA(3), CYA(3), CZA(3)	Real	Read	Arrays containing 3D WC points defining the cell array parallelogram
DIMX, DIMY	Integer	Read	Dimensions of color index array
ISC	Integer	Read	Start column in color index array
ISR	Integer	Read	Start row in color index array
DX	Integer	Read	Number of columns in color index array to be used
DY	Integer	Read	Number of rows in color index array to be used
COLIA (DIMX,DIMY)	Integer	Read	Color index array

Description

The CELL ARRAY 3 function divides a designated parallelogram into cells and displays each cell in a specified color.

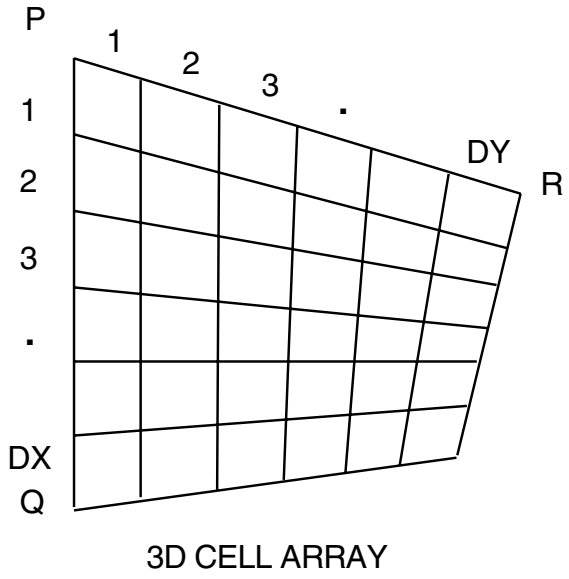
You pass a two-dimensional array containing color index values as one argument to this function. DEC GKS maps the color index values to corresponding cells within a parallelogram-shaped area of the workstation surface. In addition to the color index array, you specify an offset into the color array (a starting element), and the number of array columns and 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 parallelogram. Each of the columns within the parallelogram is of equal width, and each of the rows within the parallelogram 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. The grid defined by the three parallelogram vertices and *DIMX* and *DIMY* is subject to all transformations.

For more information on the initial color index values for a given workstation, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

To alter the color associated with a certain index value, you can use the GKS function SET COLOUR REPRESENTATION.

The following figure illustrates a transformed cell array 3 primitive. The label DX refers to the argumetn DIMX, and DY refers to the argument DIMY.



3D CELL ARRAY

ZK–4582A–GE

See Also

SET COLOUR REPRESENTATION
Example 5–1 for a program example using the CELL ARRAY function

FILL AREA

Operating States

WSAC, SGOP

Syntax

GFA (N, PXA, PYA)

Argument	Data Type	Access	Description
N	Integer	Read	Number of points in the polygon.
PXA(N), PYA(N)	Real	Read	Arrays of WC values. The number of elements in each array must be equal to the value of *N*.

Description

The FILL AREA function draws a polygon and fills it with an interior style that has already been selected. If you do not specify a closed polygon, DEC GKS connects the last point specified to the first point.

The fill area interior style can be either hollow, solid, hatched, or patterned. For example, the default fill area interior style for most supported workstation types is hollow. In that case, the function draws the outline of the polygon, leaving the interior hollow.

See Also

SET FILL AREA COLOUR INDEX
SET FILL AREA INTERIOR STYLE
SET FILL AREA STYLE INDEX
SET PATTERN REFERENCE POINT
SET PATTERN REPRESENTATION
SET PICK IDENTIFIER
Example 6–1 for a program example using the FILL AREA function

FILL AREA 3

Operating States

WSAC, SGOP

Syntax

GFA3 (N, PXA, PYA, PZA)

Argument	Data Type	Access	Description
N	Integer	Read	Number of points in the polygon.
PXA(N), PYA(N), PZA(N)	Real	Read	Arrays of 3D WC values. The number of elements in each array must be equal to the value of *N*.

Description

The FILL AREA 3 function draws a three-dimensional polygon and fills it with an interior style that has already been selected. If you do not specify a closed polygon, DEC GKS connects the last point specified to the first point.

The fill area interior style can be either hollow, solid, hatched, or patterned. For example, the default fill area interior style for most supported workstation types is hollow. In that case, the function draws the outline of the polygon, leaving the interior hollow.

See Also

SET FILL AREA COLOUR INDEX
SET FILL AREA INTERIOR STYLE
SET FILL AREA STYLE INDEX
SET PATTERN REFERENCE POINT
SET PATTERN REPRESENTATION
SET PICK IDENTIFIER
Example 6–1 for a program example using the FILL AREA function

FILL AREA SET

Operating States

WSAC, SGOP

Syntax

GFAS (NPL, IXA, XA, YA)

Argument	Data Type	Access	Description
NPL	Integer	Read	Number of lists of points.
IXA(NPL)	Integer	Read	Array of ending indexes for the point lists. The range of indexes in *XA* and *YA* for the first fill area is from 1 to *IXA(1)*. In general, the range of indexes in *XA* and *YA* for the *n*th point list is from *IXA(n–1) + 1* to *IXA(n)*.
XA(*), YA(*)	Real	Read	Arrays of coordinates of 2D WC points bounding areas.

Description

The FILL AREA SET function draws a set of two-dimensional polygons and fills them with an interior style. The fill area interior style can be either hollow, solid, hatched, or patterned. For example, the default fill area interior style for most supported workstation types is hollow. In that case, the function draws the outline of the polygons, leaving the interiors hollow.

FILL AREA SET 3

Operating States

WSAC, SGOP

Syntax

GFAS3 (NPL, IXA, XA, YA, ZA)

Argument	Data Type	Access	Description
NPL	Integer	Read	Number of lists of points.
IXA(NPL)	Integer	Read	Array of ending indexes for the point lists. The range of indexes in *XA*, *YA*, and *ZA* for the first fill area is from 1 to *IXA(1)*. In general, the range of indexes in *XA*, *YA*, and *ZA* for the *n*th point list is from *IXA(n–1) + 1* to *IXA(n)*.
XA(*), YA(*), ZA(*)	Real	Read	Arrays of coordinates of 3D WC points bounding areas.

Description

The FILL AREA SET 3 function draws a set of three-dimensional polygons and fills them with an interior style. The fill area interior style can be either hollow, solid, hatched, or patterned. For example, the default fill area interior style for most supported workstation types is hollow. In that case, the function draws the outline of the polygons, leaving the interiors hollow.

GENERALIZED DRAWING PRIMITIVE

Operating States

WSAC, SGOP

Syntax

GGDP (N, PXA, PYA, PRIMID, LDR, DATREC)

Argument	Data Type	Access	Description
N	Integer	Read	Number of points in the GDP.
PXA(N), PYA(N)	Real	Read	Arrays of X, Y values of the WC points.
PRIMID	Integer (constant)	Read	GDP identifier.
LDR	Integer	Read	Number of elements in GDP data record.
DATREC(LDR)	Character*80	Read	GDP data record. Use the PACK DATA RECORD function to construct this data record.

Constants

GDP Identifier	Description
GGDISP	Disjoint Polyline
GGCCP	Circle: Center and Point on Circumference
GGC3P	Circle: Three Points on Circumference
GGCCR	Circle: Center and Radius
GGC2PR	Circle: Two Points on Circumference, and Radius
GGAC2P	Arc: Center and Two Points on Arc
GGA3P	Arc: Three Points on Circumference
GGACVR	Arc: Center, Two Vectors, and a Radius
GGA2PR	Arc: Two Points on Arc and Radius
GGACPA	Arc: Center, Starting Point, and Angle
GGECA	Ellipse: Center, and Two Axis Vectors
GGEFP	Ellipse: Focal Points and Point on Circumference
GGEACA	Elliptic Arc: Center, Two Axis Vectors, and Two Vectors
GGEAFP	Elliptic Arc: Focal Points and Two Points on Circumference
GGR2P	Rectangle: Two Corners
GGFAS	Fill Area Set

GDP Identifier	Description
GGFCCP	Filled Circle: Center and Point on Circumference
GGFC3P	Filled Circle: Three Points on Circumference
GGFCCR	Filled Circle: Center and Radius
GGFCPR	Filled Circle: Two Points on Circumference, and Radius
GGFACP	Filled Arc: Center and Two Points on Arc
GGFA3P	Filled Arc: Three Points on Circumference
GGFACV	Filled Arc: Center, Two Vectors, and a Radius
GGFAPR	Filled Arc: Two Points on Arc, and Radius
GGFACA	Filled Arc: Center, Starting Point, and Angle
GGFECA	Filled Ellipse: Center and Two Axis Vectors
GGFEFP	Filled Ellipse: Focal Points and Point on Circumference
GGFEAC	Filled Elliptic Arc: Center, Two Axis Vectors, and Two Vectors
GGFEAF	Filled Elliptic Arc: Focal Points and Two Points on Circumference
GGFR2P	Filled Rectangle: Two Corners
GGIA	Packed Cell Array

Description

The GENERALIZED DRAWING PRIMITIVE function generates a generalized drawing primitive (GDP) of the type you specify, using specified 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 example, using DEC GKS, you can pass a center WC point and a perimeter WC 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 example, 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 more information on GDPs, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

PACK DATA RECORD
UNPACK DATA RECORD
Example 5–2 for a program example using the GENERALIZED DRAWING PRIMITIVE function

GENERALIZED DRAWING PRIMITIVE 3

Operating States

WSAC, SGOP

Syntax

GGDP3 (N, PXA, PYA, PZA, PRIMID, LDR, DATREC)

Argument	Data Type	Access	Description
N	Integer	Read	Number of points in the GDP.
PXA(N), PYA(N), PZA(N)	Real	Read	X, Y, Z values of the WC points.
PRIMID	Integer	Read	GDP identifier.
LDR	Integer	Read	Number of elements in GDP data record.
DATREC(LDR)	Character*80	Read	GDP data records. Use the PACK DATA RECORD function to construct this data record.

Description

The GENERALIZED DRAWING PRIMITIVE 3 function generates a generalized drawing primitive (GDP) of the type you specify, using specified 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 example, using DEC GKS, you can pass a center WC point and a perimeter WC 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 example, 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.

Note

Three-dimensional GDPs are not currently supported. If you call this function, DEC GKS generates an error message.

See Also

PACK DATA RECORD
UNPACK DATA RECORD
Example 5–2 for a program example using the GENERALIZED DRAWING PRIMITIVE function

POLYLINE

Operating States

WSAC, SGOP

Syntax

GPL (N, PXA, PYA)

Argument	Data Type	Access	Description
N	Integer	Read	Number of points in the polyline.
PXA(N), PYA(N)	Real	Read	X and Y WC points to be connected. The number of array elements must be equal to the value of *N*.

Description

The POLYLINE function draws one or more straight lines, connecting the WC points passed to this function in the order specified. By default, this function draws line segments as solid lines, at the nominal width, in the foreground color.

See Also

SET LINETYPE
SET LINEWIDTH SCALE FACTOR
SET PICK IDENTIFIER
SET POLYLINE COLOUR INDEX
Example 6–3 for a program example using the POLYLINE function

POLYLINE 3

Operating States

WSAC, SGOP

Syntax

GPL3 (N, PXA, PYA, PZA)

Argument	Data Type	Access	Description
N	Integer	Read	Number of points in the polyline.
PXA(N), PYA(N), PZA(N)	Real	Read	X, Y, and Z WC points to be connected. The number of array elements must be equal to the value of *N*.

Description

The POLYLINE 3 function draws one or more straight lines, connecting the three-dimensional WC points passed to this function in the order specified. By default, this function draws line segments as solid lines, at the nominal width, in the foreground color.

See Also

SET LINETYPE
SET LINEWIDTH SCALE FACTOR
SET PICK IDENTIFIER
SET POLYLINE COLOUR INDEX
Example 6–3 for a program example using the POLYLINE function

POLYMARKER

Operating States

WSAC, SGOP

Syntax

GPM (N, PXA, PYA)

Argument	Data Type	Access	Description
N	Integer	Read	Number of points to be marked.
PXA(N), PYA(N)	Real	Read	X and Y WC points of the polymarker. The number of array elements must be equal to the value of *N*.

Description

The POLYMARKER function places one or more special symbols called polymarkers at the specified WC points. By default, this function produces an asterisk polymarker, at the nominal size, in the workstation-specific default foreground color.

If clipping is enabled, and if the polymarker coordinate point is outside of the clipping rectangle, DEC GKS clips the entire polymarker. If clipping is enabled, if the polymarker coordinate point is inside of the clipping rectangle, and if portions of the polymarker exceed the boundaries of the clipping rectangle, the extent of the clipping is device dependent.

See Also

SET MARKER SIZE SCALE FACTOR
SET MARKER TYPE
SET PICK IDENTIFIER
SET POLYMARKER COLOUR INDEX
Example 6–4 for a program example using the POLYMARKER function

POLYMARKER 3

Operating States

WSAC, SGOP

Syntax

GPM3 (N, PXA, PYA, PZA)

Argument	Data Type	Access	Description
N	Integer	Read	Number of points to be marked.
PXA(N), PYA(N), PZA(N)	Real	Read	X, Y, and Z WC points of the polymarker. The number of array elements must be equal to the value of *N*.

Description

The POLYMARKER 3 function places one or more special symbols called polymarkers at the specified three-dimensional WC points. By default, this function produces an asterisk polymarker, at the the nominal size, in the workstation-specific default foreground color.

If clipping is enabled, and if the polymarker coordinate point is outside the clipping rectangle, DEC GKS clips the entire polymarker. If clipping is enabled, if the polymarker coordinate point is inside the clipping rectangle, and if portions of the polymarker exceed the boundaries of the clipping rectangle, the extent of the clipping is device dependent.

See Also

SET MARKER SIZE SCALE FACTOR
SET MARKER TYPE
SET PICK IDENTIFIER
SET POLYMARKER COLOUR INDEX
Example 6–4 for a program example using the POLYMARKER function

TEXT

Operating States

WSAC, SGOP

Syntax

GTX (PX, PY, CHARS)

Argument	Data Type	Access	Description
PX, PY	Real	Read	WC values of the point used to position the text string
CHARS	Character*(*)	Read	Text string to be written to the workstation

Description

The TEXT function writes a character string that DEC GKS positions according to the specified WC point and the current text attributes.

Depending on the current text attributes, DEC GKS positions the first character, the last character, or the middle of the text string at this WC 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.

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).

The portion of the string that DEC GKS clips depends on both the current text attributes and the workstation capabilities as follows:

- String precision: The string is clipped in a workstation-dependent manner.
- Character precision: The string is clipped character by character.
- Stroke precision: The string is clipped exactly at the normalization viewport.

See Also

SET CHARACTER EXPANSION FACTOR
SET CHARACTER HEIGHT
SET CHARACTER SPACING
SET CHARACTER UP VECTOR
SET PICK IDENTIFIER
SET TEXT ALIGNMENT
SET TEXT COLOUR INDEX
SET TEXT FONT AND PRECISION
SET TEXT PATH
Example 6–4 for a program example using the TEXT function

TEXT (FORTRAN–77 Subset)

Operating States

WSAC, SGOP

Syntax

GTXS (PX, PY, LSTR, CHARS)

Argument	Data Type	Access	Description
PX, PY	Real	Read	WC values of the point used to position the text string
LSTR	Integer	Read	Length of the string in characters
CHARS	Character*80	Read	Text string to be written to the workstation

Description

The TEXT function writes a character string that DEC GKS positions according to the specified WC point and the current text attributes.

Depending on the current text attributes, DEC GKS positions the first character, the last character, or the middle of the text string at this WC 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.

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).

The portion of the string that DEC GKS clips depends on both the current text attributes and the workstation capabilities as follows:

- String precision: The string is clipped in a workstation-dependent manner.
- Character precision: The string is clipped character by character.
- Stroke precision: The string is clipped exactly at the normalization viewport.

See Also

SET CHARACTER EXPANSION FACTOR
SET CHARACTER HEIGHT
SET CHARACTER SPACING
SET CHARACTER UP VECTOR
SET PICK IDENTIFIER
SET TEXT ALIGNMENT
SET TEXT COLOUR INDEX
SET TEXT FONT AND PRECISION
SET TEXT PATH
Example 6–4 for a program example using the TEXT function

TEXT 3

Operating States

WSAC, SGOP

Syntax

GTX3 (PX, PY, PZ, TDX, TDY, TDZ, CHARS)

Argument	Data Type	Access	Description
PX, PY, PZ	Real	Read	WC values of the point used to position the text string.
TDX(2), TDY(2), TDZ(2)	Real	Read	Two direction vectors, which together with the point (PX, PY, PZ) define the text plane. If U and V are the first and second direction vectors respectively, the text plane is the plane that contains point (PX, PY, PZ) and is perpendicular to U x V.
CHARS	Character*(*)	Read	Text string to be written to the workstation.

Description

The TEXT 3 function writes a character string that DEC GKS positions according to the specified WC point and the current text attributes.

Depending on the current text attributes, DEC GKS positions the first character, the last character, or the middle of the text string at this WC 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.

The orientation of the characters is given by text direction vectors. The shape of the characters depends on the current text attributes, the current normalization transformation, and the workstation capabilities.

The portion of the string that DEC GKS clips depends on both the current text attributes and the workstation capabilities as follows:

- String precision: The string is clipped in a workstation-dependent manner.
- Character precision: The string is clipped character by character.
- Stroke precision: The string is clipped exactly at the normalization viewport.

See Also

SET CHARACTER EXPANSION FACTOR
SET CHARACTER HEIGHT
SET CHARACTER SPACING
SET CHARACTER UP VECTOR
SET PICK IDENTIFIER
SET TEXT ALIGNMENT
SET TEXT COLOUR INDEX
SET TEXT FONT AND PRECISION
SET TEXT PATH
Example 6–4 for a program example using the TEXT function

TEXT 3 (FORTRAN–77 Subset)

Operating States

WSAC, SGOP

Syntax

GTX3S (PX, PY, PZ, TDX, TDY, TDZ, NCHS, CHARS)

Argument	Data Type	Access	Description
PX, PY, PZ	Real	Read	WC values of the point used to position the text string.
TDX(2), TDY(2), TDZ(2)	Real	Read	Two direction vectors, which together with the point (PX, PY, PZ) define the text plane. If U and V are the first and second direction vectors respectively, the text plane is the plane that contains point (PX, PY, PZ) and is perpendicular to U x V.
NCHS	Integer	Read	Number of characters in string.
CHARS	Character*(*)	Read	Text string to be written to the workstation.

Description

The TEXT 3 function writes a character string that DEC GKS positions according to the specified WC point and the current text attributes.

Depending on the current text attributes, DEC GKS positions the first character, the last character, or the middle of the text string at this WC 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.

The orientation of the characters is given by text direction vectors. The shape of the characters depends on the current text attributes, the current normalization transformation, and the workstation capabilities.

The portion of the string that DEC GKS clips depends on both the current text attributes and the workstation capabilities as follows:

- String precision: The string is clipped in a workstation-dependent manner.
- Character precision: The string is clipped character by character.
- Stroke precision: The string is clipped exactly at the normalization viewport.

See Also

SET CHARACTER EXPANSION FACTOR
SET CHARACTER HEIGHT
SET CHARACTER SPACING
SET CHARACTER UP VECTOR
SET PICK IDENTIFIER
SET TEXT ALIGNMENT
SET TEXT COLOUR INDEX
SET TEXT FONT AND PRECISION
SET TEXT PATH
Example 6–4 for a program example using the TEXT function

5.7 Program Examples

Example 5–1 illustrates the use of the CELL ARRAY function.

Example 5–1 Cell Array Output

```
C  This code example draws alternating white and black vertical stripes
C  using the CELL ARRAY function.
C
C  NOTE: To keep the example concise, no error checking is performed.

      IMPLICIT NONE
      INCLUDE  'gks.f'

      INTEGER   COLIA(10)
      INTEGER   DEVNUM
      INTEGER   DIMX
      INTEGER   DIMY
      INTEGER   DX
      INTEGER   DY
      INTEGER   ICL
      INTEGER   ISC
      INTEGER   ISR
      INTEGER   NUMCOL
      INTEGER   NUMROW
      REAL      PX
      REAL      PY
      REAL      QX
      REAL      QY
      REAL      TIME
      INTEGER   WS_ID

C  Open the GKS and workstation environments.

      WS_ID = 1

      CALL GOPKS (0, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)
      CALL GSDS (WS_ID, GASTI, GSUPPD)

C  Draw the stripes. Wait 5 seconds.
```

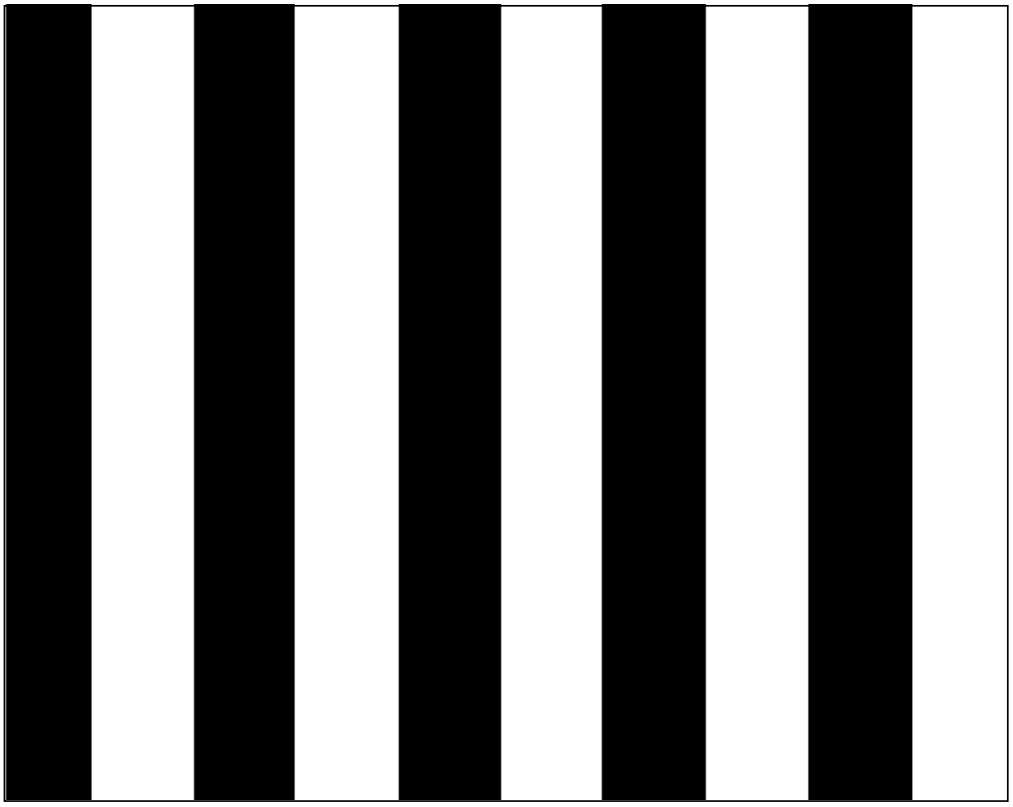
(continued on next page)

Example 5–1 (Cont.) Cell Array Output

```
      COLIA(1)  = 0
      COLIA(2)  = 1
      COLIA(3)  = 0
      COLIA(4)  = 1
      COLIA(5)  = 0
      COLIA(6)  = 1
      COLIA(7)  = 0
      COLIA(8)  = 1
      COLIA(9)  = 0
      COLIA(10) = 1
      DX = 10
      DY = 1
      DIMX = 10
      DIMY = 1
      ISC = 1
      ISR = 1
      NUMCOL = 10
      NUMROW = 1
      PX = 0.0
      PY = 0.0
      QX = 1.0
      QY = 1.0
      TIME = 5.0

      CALL GCA (PX, PY, QX, QY, DIMX, DIMY, ISC, ISR, DX, DY, COLIA)
      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIME, WS_ID, ICL, DEVNUM)

C  Release the GKS and workstation environments.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 5–1 shows the program's effect on a VAXstation workstation running DECwindows software.

Figure 5–1 Cell Array Output



ZK–4015A–GE

Example 5–2 illustrates the use of the GENERALIZED DRAWING PRIMITIVE function.

Example 5–2 Generalized Drawing Primitive Output

```
C  This program creates an unfilled circle using the GDP
C  GGC3P (-102).

      IMPLICIT NONE
      INCLUDE  'gks.f'

      CHARACTER*80 DATA_RECORD
      INTEGER      DEVNUM
      INTEGER      ICL
      INTEGER      NUM_POINTS
      REAL         PX (3)
      REAL         PY (3)
      INTEGER      RECORD_SIZE
      REAL         TIME
      INTEGER      WS_ID
```

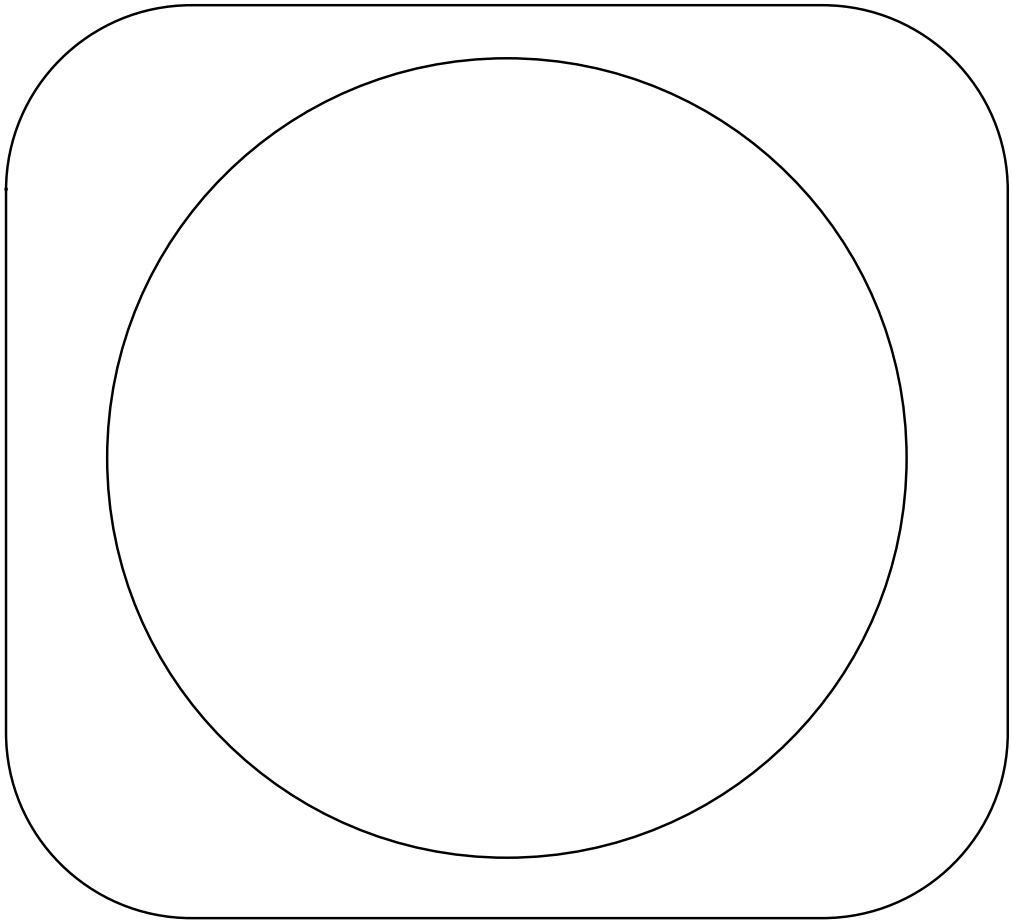
(continued on next page)

Example 5–2 (Cont.) Generalized Drawing Primitive Output

```
C  Open the GKS and workstation environments.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Specify the points that define the circle.

      PX(1) = 0.1
      PY(1) = 0.5

      PX(2) = 0.5
      PY(2) = 0.1

      PX(3) = 0.9
      PY(3) = 0.5

C  The constant GGC3P specifies the GDP identification number -102.
C  This GDP creates a circle using three points on a circle's
C  circumference.  This particular GDP does not require a data record
C  to perform its task.  Notice that DEC GKS uses the current polyline
C  attributes to create the circle.

      NUM_POINTS = 3
      RECORD_SIZE = 0
      TIME = 5.00

      CALL GGDP (NUM_POINTS, PX, PY, GGC3P,
     *RECORD_SIZE, DATA_RECORD)
      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIME, WS_ID, ICL, DEVNUM)

C  Release the GKS and workstation environments.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 5–2 shows the program's effect on a VAXstation workstation running DECwindows software.

Figure 5–2 Generalized Drawing Primitive Output



ZK–4013A–GE

Attribute Functions

Insert tabbed divider here. Then discard this sheet.

6
Attribute Functions

The DEC GKS attribute functions affect the appearance of generated output primitives.

The GKS state list stores the current value of the attributes for each output function. These attributes specify the exact appearance of the object drawn. For example, when you call POLYLINE, the attributes line type, width scale factor, and color specify the form, thickness, and color of the line. In the 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 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**, **viewing**, and **pick identification** aspects of a graphic image. The geometric, nongeometric, and viewing aspects of a graphic image directly affect how the primitive appears on the workstation surface. The viewing attributes are the view index and the HLHSR identifier. The view index is a pointer to a workstation view table entry. The HLHSR identifier provides hidden line and hidden surface information about the primitive. For more information on viewing, see Chapter 7. The pick identification attribute is used to identify a primitive, or group of primitives, in a segment when that segment is picked. For complete details concerning pick input, see Chapter 9.

Most output functions have nongeometric attributes that are changeable. Nongeometric attributes affect the style and the pattern of the output primitives (such as polyline color, text spacing, and fill area interior style). Because 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, fill area, and fill area set primitives (such as text height, character path, and pattern size). Text, fill area, and fill area set are the only output primitives that have changeable geometric attributes. The geometric attributes are specified in world coordinate (WC) units. Therefore, because the WC units are device independent, the geometric attributes are device independent.

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

Table 6–1 Geometric and Nongeometric 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 scale factor	Nongeometric
	Polymarker color index	Nongeometric
Text	Text index	Nongeometric
	Text font and precision	Nongeometric
	Character expansion factor	Nongeometric
	Character spacing	Nongeometric
	Text color index	Nongeometric
	Character height	Geometric
	Character up vector	Geometric
	Text path	Geometric
	Text alignment	Geometric
Fill area	Fill area index	Nongeometric
	Fill area interior style	Nongeometric
	Fill area style index	Nongeometric
	Fill area color index	Nongeometric
	Pattern size	Geometric
	Pattern reference point and vectors	Geometric

(continued on next page)

Table 6–1 (Cont.) Geometric and Nongeometric Attributes

Function	Attribute	Type
Fill area set	Fill area index	Nongeometric
	Fill area interior style	Nongeometric
	Fill area style index	Nongeometric
	Fill area color index	Nongeometric
	Edge index	Nongeometric
	Edge flag	Nongeometric
	Edge type	Nongeometric
	Edge width scale factor	Nongeometric
	Edge color index	Nongeometric
	Pattern size	Geometric
	Pattern reference point and vectors	Geometric

Notice that there are no geometric or nongeometric 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 geometric or nongeometric 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, fill area, or fill area set attribute values, depending on the nature of the GDP.

6.2 Individual and Bundled Attribute Values

The current values of each attribute are listed individually in the GKS state list. By default, a call to an output function uses these individual attribute values to generate the primitive. Because DEC GKS stores these individual attributes in the GKS state list, they are device independent. If you specify attributes individually, you cannot change a primitive's appearance on the workstation surface once you have generated 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 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 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 can 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, see Section 6.2.2.

To review the initial individual attributes and the bundle tables available on a given workstation, see Appendix E.

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 13-element integer array, one element for every nongeometric attribute. Each element contains either the value GBUNDL (0) or the value GINDIV (1). By passing this array to the function SET ASPECT SOURCE FLAGS, DEC GKS uses either the individual attribute value or the bundled value in the bundle table specified by the current bundle index.

For a complete description of ASFs, see the SET ASPECT SOURCE FLAGS 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 GBUNDL 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, 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 (SET POLYLINE REPRESENTATION, SET POLYMARKER REPRESENTATION, and so on).

If you use the SET . . . 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 because of the loss of all primitives not contained in segments.

For a detailed description of implicit regeneration, see Chapter 4.

6.3 Foreground and Background Colors

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

On an OUTIN or 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 SET COLOUR REPRESENTATION, the color index value 0 corresponds to the workstation's 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 Attribute Inquiries

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

INQUIRE COLOUR FACILITIES
INQUIRE COLOUR REPRESENTATION
INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES (3)
INQUIRE CURRENT PRIMITIVE ATTRIBUTE VALUES (3)
INQUIRE FILL AREA FACILITIES
INQUIRE FILL AREA REPRESENTATION
INQUIRE LIST OF COLOUR INDICES
INQUIRE LIST OF FILL AREA INDICES
INQUIRE LIST OF PATTERN INDICES
INQUIRE LIST OF POLYLINE INDICES
INQUIRE LIST OF POLYMARKER INDICES
INQUIRE LIST OF TEXT INDICES
INQUIRE MAXIMUM LENGTH OF WORKSTATION STATE TABLES
INQUIRE PATTERN FACILITIES
INQUIRE PATTERN REPRESENTATION
INQUIRE POLYLINE FACILITIES
INQUIRE POLYLINE REPRESENTATION
INQUIRE POLYMARKER FACILITIES
INQUIRE POLYMARKER REPRESENTATION
INQUIRE PREDEFINED COLOUR REPRESENTATION
INQUIRE PREDEFINED EDGE REPRESENTATION
INQUIRE PREDEFINED FILL AREA REPRESENTATION
INQUIRE PREDEFINED PATTERN REPRESENTATION
INQUIRE PREDEFINED POLYLINE REPRESENTATION
INQUIRE PREDEFINED POLYMARKER REPRESENTATION
INQUIRE PREDEFINED TEXT REPRESENTATION
INQUIRE SET OF OPEN WORKSTATIONS
INQUIRE TEXT FACILITIES
INQUIRE TEXT REPRESENTATION

For more information concerning device-independent programming, see the *DEC GKS User's Guide*.

6.5 Function Descriptions

This section describes the DEC GKS attribute functions in detail.

SET ASPECT SOURCE FLAGS

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSASF (LASF)

Argument	Data Type	Access	Description
LASF(13)	Integer (constant)	Read	Array of 13 ASFs

Constants

Defined Argument	Constant	Description
LASF	GBUNDL	Bundled attributes
	GINDIV	Individual attributes

Description

The SET ASPECT SOURCE FLAGS function specifies to DEC GKS whether to use the bundled or the individual method for designating each of the nongeometric output attributes.

There are 13 nongeometric ASFs. These flags are as follows:

1. Polyline type
2. Polyline width
3. Polyline color
4. Polymarker type
5. Polymarker size
6. Polymarker color
7. Text font and precision
8. Character expansion
9. Character spacing
10. Text color
11. Fill area interior style
12. Fill area style index
13. Fill area color

If the value in the corresponding element is GINDIV, DEC GKS uses the individual attribute setting. If the value in the corresponding element is GBUNDL, DEC GKS uses the bundle table index to find the attribute setting.

The initial value for each ASF is GINDIV, 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 example, 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, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

SET FILL AREA INDEX
SET POLYLINE INDEX
SET POLYMARKER INDEX
SET TEXT INDEX
Example 6–2 for a program example using the SET ASPECT SOURCE FLAGS function

SET ASPECT SOURCE FLAGS 3

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSASF3 (ASF)

Argument	Data Type	Access	Description
ASF(17)	Integer (constant)	Read	Array of 17 ASFs

Constants

Defined Argument	Constant	Description
LASF	GBUNDL	Bundled attributes
	GINDIV	Individual attributes

Description

The SET ASPECT SOURCE FLAGS 3 function specifies to DEC GKS whether to use the bundled or the individual method for designating each of the nongeometric output attributes.

There are 17 nongeometric ASFs. These flags are as follows:

1. Polyline type
2. Polyline width
3. Polyline color
4. Polymarker type
5. Polymarker size
6. Polymarker color
7. Text font and precision
8. Character expansion
9. Character spacing
10. Text color
11. Fill area interior style
12. Fill area style index
13. Fill area color
14. Edge flag

15. Edge type
16. Edge width
17. Edge color

If the value in the corresponding element is GINDIV, DEC GKS uses the individual attribute setting. If the value in the corresponding element is GBUNDL, DEC GKS uses the bundle table index to find the attribute setting.

The initial value for each ASF is GINDIV, 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 example, 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, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

SET FILL AREA INDEX
SET POLYLINE INDEX
SET POLYMARKER INDEX
SET TEXT INDEX
Example 6–2 for a program example using the SET ASPECT SOURCE FLAGS function

SET CHARACTER EXPANSION FACTOR

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSCHXP (CHXP)

Argument	Data Type	Access	Description
CHXP	Real	Read	Character expansion factor

Description

The SET CHARACTER EXPANSION FACTOR function sets the *current character expansion factor* entry in the GKS state list to the specified value. This function alters the width of the generated characters, but not the height. The character expansion factor is multiplied by the width-to-height ratio specified in the original font specification to give the new character width.

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.

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 use either the SET CHARACTER EXPANSION FACTOR function or the SET CHARACTER HEIGHT function to reestablish a legible character width.

See Also

SET ASPECT SOURCE FLAGS
SET CHARACTER HEIGHT
SET CHARACTER SPACING
SET TEXT INDEX
SET TEXT REPRESENTATION
TEXT

SET CHARACTER HEIGHT

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSCHH (CHH)

Argument	Data Type	Access	Description
CHH	Real	Read	Character height in WC units. The default value is 0.01.

Description

The SET CHARACTER HEIGHT function sets the geometric attribute, *current character height* entry in the GKS state list to the specified WC unit value.

DEC GKS uses the value specified in the call to SET CHARACTER HEIGHT for all subsequent calls to TEXT until you specify another value. If you specify a new height to this function, DEC GKS expands text output to the closest height the workstation is capable of producing. The default for the current text height is the WC unit value 0.01. This is 0.01 of the default normalization window height (1.0). Exercise caution if you change the size of the current normalization window, as 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.

See Also

SELECT NORMALIZATION TRANSFORMATION
SET WINDOW
Example 6–4 for a program example using the SET CHARACTER HEIGHT function

SET CHARACTER SPACING

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSCHSP (CHSP)

Argument	Data Type	Access	Description
CHSP	Real	Read	Spacing as a percentage of character height. The default value is 0.0, which displays text with adjacent character bodies.

Description

The SET CHARACTER SPACING function sets the *current text spacing* entry in the 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 adjusts spacing proportionately. 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 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.

See Also

SET ASPECT SOURCE FLAGS
SET TEXT FONT AND PRECISION
SET TEXT INDEX
SET TEXT REPRESENTATION
TEXT

SET CHARACTER UP VECTOR

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSCHUP (CHUX, CHUY)

Argument	Data Type	Access	Description
CHUX, CHUY	Real	Read	X and Y values that specify the slope of the character up vector. The default value is (0.0, 1.0).

Description

The SET CHARACTER UP VECTOR function sets the geometric attribute, *current character up vector* entry in the GKS state list to the specified value.

DEC GKS uses the value specified in the call to SET CHARACTER UP VECTOR for all subsequent calls to TEXT until you specify another value. When you call TEXT, you specify the starting point for the text. 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 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 positioning 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 WC 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).

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 RIGHT or LEFT.

See Also

SET CHARACTER HEIGHT
SET TEXT PATH

SET COLOUR MODEL

Operating States

WSOP, WSAC, SGOP

Syntax

GSCMD (WKID, CMODEL)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
CMODEL	Integer (constant)	Read	Color model value

Constants

Defined Argument	Constant	Description
CMODEL	GRGB	Red, green, and blue color model
	GHCIE	Commission Internationale de l'Eclairage color model
	GHSV	Hue, saturation, and value color model
	GHLS	Hue, lightness, and saturation color model

Description

The SET COLOUR MODEL function sets the color model of the specified workstation to the specified model value.

This function implicitly regenerates the workstation surface. Implicit regeneration is described in the *DEC GKS User's Guide*.

Attribute values passed to this function must be valid for the specified workstation. For more information concerning device-specific attributes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*. For a description of the color models, see the *DEC GKS User's Guide*.

SET COLOUR REPRESENTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GSCR (WKID, COLI, COMP1, COMP2, COMP3)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
COLI	Integer	Read	Color index value
COMP1, COMP2, COMP3	Real	Read	Color components corresponding to the current color model

Description

The SET COLOUR REPRESENTATION function allows the user to redefine an existing color index representation, or to define a new representation, by specifying the color triplet 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. Values greater than 1 correspond to alternative foreground colors.

There are four different color models, and the values you use for the color triplet vary according to the color model you use. The color models and their required values are as follows:

Model	Description	Values
RGB	Red intensity, green intensity, blue intensity	Each component must be in the range 0.0 to 1.0.
CIE	X and Y chromaticity coefficients, luminance value Y	Each component must be in the range 0.0 to 1.0.
HSV	Hue, saturation, and value	The hue component must be in the range 0.0 to 360.0, and the saturation and value components must be in the range 0.0 to 1.0.
HLS	Hue, lightness, and saturation	The hue component must be in the range 0.0 to 360.0, and the saturation and value components must be in the range 0.0 to 1.0.

Depending on the capabilities of your workstation, a call to this function may cause DEC GKS to implicitly regenerate the workstation surface.

Attribute values passed to this function must be valid for the specified workstation. For more information concerning device-specific attributes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

INQUIRE COLOUR FACILITIES
INQUIRE COLOUR REPRESENTATION
Example 6–1 for a program example using the SET COLOUR REPRESENTATION function

SET EDGE COLOUR INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSEDCI (COLI)

Argument	Data Type	Access	Description
COLI	Integer	Read	Edge color index

Description

The SET EDGE COLOUR INDEX function selects the *current edge color index* entry in the GKS state list to the specified value. This value controls the display of subsequent fill area set output primitives when the current edge color index ASF has been set to INDIVIDUAL by the function SET ASPECT SOURCE FLAGS 3. If this ASF is set to BUNDLED, the current edge flag has no effect.

The edge color index points into the color tables of the workstation and is a positive integer. If the specified color index is not present in a workstation color table, a workstation-dependent color index is used.

See Also

Example 6–4 for a program example using a SET . . . COLOUR INDEX function

SET EDGE FLAG

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSEDFG (EDFLAG)

Argument	Data Type	Access	Description
EDFLAG	Integer (constant)	Read	Edge flag

Constants

Defined Argument	Constant	Description
EDFLAG	GOFF	Edge off. This is the default value.
	GON	Edge on.

Description

The SET EDGE FLAG function sets or resets the *current edge flag* entry in the GKS state list to the specified value.

The current edge flag enables the display of subsequent fill area set output primitives when the current edge flag ASF has been set to INDIVIDUAL by the function SET ASPECT SOURCE FLAGS 3. If this ASF is set to BUNDLED, the current edge flag has no effect.

SET EDGE INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSEDI (EDI)

Argument	Data Type	Access	Description
EDI	Integer	Read	Edge index. The default value is 1.

Description

The SET EDGE INDEX function sets the *current edge index* entry in the GKS state list to the specified index value. The edge bundle table contains entries for the attribute values, edge flag, edge type, edge width scale factor, and edge color index.

SET EDGE INDEX controls which edge bundle table entry will be used when subsequent fill area set primitives are generated. Attribute values are taken from the edge bundle table only if the edge ASFs were set to BUNDLED by the function SET ASPECT SOURCE FLAG 3.

See Also

Example 6–2 for a program example using a SET . . . INDEX function

SET EDGE REPRESENTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GSEDR (WKID, EDI, EDFLAG, EDTYPE, EDWSF, COLI)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
EDI	Integer	Read	Edge bundle table index value
EDFLAG	Integer (constant)	Read	Edge flag
EDTYPE	Integer (constant)	Read	Edge type
EDWSF	Real	Read	Edge width scale factor
COLI	Integer	Read	Edge color index

Constants

Defined Argument	Constant	Description
EDFLAG	GOFF	Edge off. This is the default value.
	GON	Edge on.
EDTYPE	GLSOL	Solid edge. This is the default value.
	GLDASH	Dashed edge.
	GLDOT	Dotted edge.
	GLDASD	Dashed-dotted edge.

Note

Other, nonstandard, edge types are available. See Appendix B.

Description

The SET EDGE REPRESENTATION function allows the user to redefine the representation of an existing edge bundle table index, or to define a new edge bundle table index value. If fill area sets are displayed with an edge index value that is not in the edge bundle table, edge index 1 is used.

This function implicitly regenerates the workstation surface if the workstation is capable of implicit regeneration. Implicit regeneration is described in the *DEC GKS User's Guide*.

Attribute values passed to this function must be valid for the specified workstation. For more information concerning device-specific attributes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

Example 6–1 for a program example using a SET . . . REPRESENTATION function

SET EDGETYPE

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSEDT (EDTYPE)

Argument	Data Type	Access	Description
EDTYPE	Integer (constant)	Read	Edge type

Constants

Defined Argument	Constant	Description
EDTYPE	GESOLI	Solid edge. This is the default value.
	GEDASH	Dashed edge.
	GEDOT	Dotted edge.
	GEDASD	Dashed-dotted edge.

Note

Other, nonstandard, edge types are available. See Appendix B.

Description

The SET EDGETYPE function sets the *current edge type* entry in the GKS state list to the specified value. The value of this entry controls the display of subsequent fill area set output primitives when the current edge type ASF has been set to INDIVIDUAL by the function SET ASPECT SOURCE FLAGS 3. If this ASF is set to BUNDLED, the current edge flag has no effect.

See Also

Example 6–3 for a program example using a SET . . . TYPE function

SET EDGEWIDTH SCALE FACTOR

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSEWSC (EDWSF)

Argument	Data Type	Access	Description
EDWSF	Real	Read	Edge width scale factor. The default value is 1.0.

Description

The SET EDGEWIDTH SCALE FACTOR function sets the *current edge width scale factor* entry in the GKS state list to the specified value. This value controls the display of subsequent fill area set output primitives when the current edge width factor ASF has been set to INDIVIDUAL by the function SET ASPECT SOURCE FLAGS 3. If this ASF is set to BUNDLED, the current edge flag has no effect.

The edge width scale factor is supplied to the nominal workstation edge width, and the result is mapped to the workstation in the nearest available edge width.

SET FILL AREA COLOUR INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSFACI (COLI)

Argument	Data Type	Access	Description
COLI	Integer	Read	Fill area color index. The default value is 1, which designates the default foreground color.

Description

The SET FILL AREA COLOUR INDEX function sets the *current fill area color index* entry in the GKS state list to the specified index value. The specified index value is used for the display of subsequent FILL AREA and FILL AREA SET output primitives, created when the current fill area color index ASF is INDIVIDUAL. This value does not affect the display of subsequent FILL AREA and FILL AREA SET output primitives, created when the current fill area color index ASF is BUNDLED.

If the specified color index is not present in a workstation color table, a workstation-dependent color index is used on that workstation.

See Also

SET ASPECT SOURCE FLAGS
SET COLOUR REPRESENTATION
SET FILL AREA INDEX
SET FILL AREA REPRESENTATION
Example 6–1 for a program example using the SET FILL AREA COLOUR INDEX function

SET FILL AREA INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSFAI (FAI)

Argument	Data Type	Access	Description
FAI	Integer	Read	Fill area bundle index. The default value is 1.

Description

The SET FILL AREA INDEX function establishes the index value pointing into the fill area bundle table. This table contains entries for the attribute values, fill area interior style, fill area style index, and fill area color index. When calling the SET FILL AREA INDEX function, DEC GKS uses the bundle table only if the corresponding ASF has been set to BUNDLED.

See Also

SET ASPECT SOURCE FLAGS
SET FILL AREA REPRESENTATION
Example 6–2 for a program example using the SET FILL AREA INDEX function

SET FILL AREA INTERIOR STYLE

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSFAIS (INTS)

Argument	Data Type	Access	Description
INTS	Integer (constant)	Read	Fill area interior style

Constants

Defined Argument	Constant	Description
INTS	GHOLLO	Hollow interior. This is the default value.
	GSOLID	Solid interior.
	GPATTR	Patterned interior.
	GHATCH	Hatched interior.

Description

The SET FILL AREA INTERIOR STYLE function sets the *current fill area interior style* entry in the GKS state list to be hollow, solid, pattern, or hatched. If you set the fill area interior style to SOLID, the FILL AREA function fills the color designated by the current fill area color index.

If you select pattern, the FILL AREA function 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 the SET PATTERN SIZE and the SET PATTERN REFERENCE POINT functions). The fill area style index specifies the pattern to replicate (see the SET FILL AREA STYLE INDEX function). Patterns cover underlying primitives.

If you select hatched, the FILL AREA function fills the interior of the polygon with a series of parallel or cross-hatch lines in the color specified by the fill area color index. The fill area style index specifies the chosen hatch style. The space between the parallel or cross-hatch lines is transparent.

See the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on the hatch patterns available on your device.

See Also

FILL AREA
SET ASPECT SOURCE FLAGS
SET FILL AREA INDEX
SET FILL AREA REPRESENTATION
SET FILL AREA STYLE INDEX
SET PATTERN REFERENCE POINT
SET PATTERN SIZE
Example 6–1 for a program example using the SET FILL AREA INTERIOR STYLE function

SET FILL AREA REPRESENTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GSFAR (WKID, FAI, INTS, STYLI, COLI)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
FAI	Integer	Read	Fill area bundle table index value.
INTS	Integer (constant)	Read	Interior style index value.
STYLI	Integer	Read	Fill area style index. If you specify GHOLLO or GSOLID for *INTS*, DEC GKS ignores the *STYLI* argument.
COLI	Integer	Read	Fill area color index.

Constants

Defined Argument	Constant	Description
INTS	GHOLLO	Hollow interior. This is the default value.
	GSOLID	Solid interior.
	GPATTR	Patterned interior.
	GHATCH	Hatched interior.

Description

The SET FILL AREA REPRESENTATION function allows the user 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 the SET FILL AREA REPRESENTATION function may cause DEC GKS to implicitly regenerate the workstation surface.

Attribute values passed to this function must be valid for the specified workstation. For more information concerning device-specific attributes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

SET ASPECT SOURCE FLAGS
SET FILL AREA INDEX
SET FILL AREA INTERIOR STYLE
Example 6–2 for a program example using the SET FILL AREA REPRESENTATION function

SET FILL AREA STYLE INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSFASI (STYLI)

Argument	Data Type	Access	Description
STYLI	Integer	Read	Fill area style index. The default value is 1.

Description

The SET FILL AREA STYLE INDEX function sets the *current fill area style index* entry in the GKS state list to the specified index value.

If the interior style is hollow or solid, the current style index is ignored for the call to 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. For device-dependent hatch styles, the hatch style index is always a negative number.

If the requested style index is not available on the specified workstation, the workstation uses the style index 1. If style index 1 is not present on the workstation, the resulting output is workstation dependent.

See Also

SET ASPECT SOURCE FLAGS
SET FILL AREA INDEX
SET FILL AREA INTERIOR STYLE
SET FILL AREA REPRESENTATION
SET PATTERN REFERENCE POINT
SET PATTERN REPRESENTATION
SET PATTERN SIZE

SET HLHSR IDENTIFIER

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSHRID (HRID)

Argument	Data Type	Access	Description
HRID	Integer (constant)	Read	HLHSR identifier

Constants

Defined Argument	Constant	Description
HRID	GHRIN	No HLHSR processing. This is the default value.
	GHRIPT	Painters algorithm.

Description

The SET HLHSR IDENTIFIER function sets the current hidden line and hidden surface removal (HLHSR) identifier to the value specified. If the requested HLHSR identifier cannot be interpreted at the workstation, the workstation uses another HLHSR identifier.

SET HLHSR MODE

Operating States

WSOP, WSAC, SGOP

Syntax

GSHRM (WKID, HRM)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
HRM	Integer (constant)	Read	HLHSR mode

Constants

Defined Argument	Constant	Description
HRM	GHRMN	No HLHSR processing. This is the default value.
	GHRMPT	Painters algorithm.

Description

The SET HLHSR MODE function sets the requested *HLHSR mode* entry in the workstation state list of the specified workstation to the specified mode value. The effect of the specified mode value is influenced by the current settings of the *dynamic modification accepted for HLHSR mode (DMA)* entry in the workstation description table and the *display surface empty (DSE)* entry in the workstation state list. If the DMA entry is IMM, or if the DSE entry is EMPTY, then the current HLHSR mode is set to the specified value, and the HLHSR update state is set to NOTPENDING. Otherwise, the current HLHSR mode is not changed, and the HLHSR update state is set to PENDING.

SET LINETYPE

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSLN (LTYPE)

Argument	Data Type	Access	Description
LTYPE	Integer (constant)	Read	Polyline type

Constants

Defined Argument	Constant	Description
LTYPE	GLSOL	Solid line. This is the default value.
	GLDASH	Dashed line.
	GLDOT	Dotted line.
	GLDASD	Dashed-dotted line.

Note

Other, nonstandard, polyline types are available. See Appendix B.

Description

The SET LINETYPE function sets the *current polyline type* entry in the GKS state list to solid, dashed, dotted, dashed-dotted, or any one of the device-dependent types.

Every workstation capable of output (DEC GKS workstation category OUTPUT or OUTIN) defines at least four line types. For more information concerning possible polyline type values, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

SET POLYLINE REPRESENTATION
Example 6–3 for a program example using the SET LINETYPE function

SET LINEWIDTH SCALE FACTOR

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSLWSC (LWIDTH)

Argument	Data Type	Access	Description
LWIDTH	Real	Read	Line width scale factor. The default value is 1.0.

Description

The SET LINEWIDTH SCALE FACTOR function sets the *current polyline width scale factor* entry in the 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 this function. The graphics handler maps the value to the nearest available line width defined by the graphics handler.

SET MARKER SIZE SCALE FACTOR

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSMKSC (MSZSF)

Argument	Data Type	Access	Description
MSZSF	Real	Read	Marker size scale factor. The default value is 1.0.

Description

The SET MARKER SIZE SCALE FACTOR function sets the *current marker size scale factor* entry in the GKS state list to the specified value for all polymarker types.

DEC GKS calculates polymarker size for all types (except the dot polymarker type) as the nominal polymarker size multiplied by the polymarker size scale factor. The polymarker size scale factor is a real number that you pass to this function. The graphics handler maps the value to the nearest available polymarker size defined by the handler. (The dot polymarker type is always the smallest dot that the workstation can produce.)

See Also

POLYMARKER
SET ASPECT SOURCE FLAGS
SET POLYMARKER INDEX
SET POLYMARKER REPRESENTATION

SET MARKER TYPE

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSMK (MTYPE)

Argument	Data Type	Access	Description
MTYPE	Integer (constant)	Read	Polymarker type

Constants

Defined Argument	Constant	Description
MTYPE	GPOINT	Dot.
	GPLUS	Plus sign.
	GAST	Asterisk. This is the default value.
	GOMARK	Circle.
	GXMARK	Diagonal cross.

Note

Other, nonstandard, polymarker types are available. See Appendix B.

Description

The SET MARKER TYPE function sets the *current marker type* entry in the GKS state list to be dot, plus sign, asterisk, circle, diagonal cross, or any of the device-dependent types.

Every workstation capable of output (DEC GKS workstation category OUTPUT or OUTIN) defines at least five polymarker types. For more information concerning predefined polymarker type values, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

POLYMARKER
SET MARKER SIZE SCALE FACTOR
SET POLYMARKER COLOUR INDEX
SET POLYMARKER INDEX
SET POLYMARKER REPRESENTATION
Example 6–4 for a program example using the SET MARKER TYPE function

SET PATTERN REFERENCE POINT

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSPARF (RFX, RFY)

Argument	Data Type	Access	Description
RFX, RFY	Real	Read	Pattern reference point in WC points

Description

The SET PATTERN REFERENCE POINT function sets the geometric attribute, *current pattern reference point* entry in the GKS state list.

The current pattern reference point attribute represents the starting point for a pattern used to fill the designated area. DEC GKS uses this value for all subsequent calls to FILL AREA until you specify another value.

Most of the DEC GKS supported workstations do not fully support this function. They do accept the function call, but do not make any changes to the pattern. For more information concerning patterns, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

FILL AREA
SET FILL AREA INTERIOR STYLE
SET FILL AREA STYLE INDEX
SET PATTERN SIZE

SET PATTERN REFERENCE POINT AND VECTORS

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSPRPV (PRPX, PRPY, PRPZ, PRVX, PRVY, PRVZ)

Argument	Data Type	Access	Description
PRPX, PRPY, PRPZ	Real	Read	Pattern reference point coordinates in WC points
PRVX(2), PRVY(2), PRVZ(2)	Real	Read	Pattern reference vectors

Description

The SET PATTERN REFERENCE POINT AND VECTORS function sets the geometric attributes, *current pattern reference points 3* and *current pattern reference vectors* entries in the GKS state list.

The current pattern reference point 3 attribute represents the starting point for a pattern used to fill the designated area. DEC GKS uses this value for all subsequent calls to the FILL AREA function until another value is specified. When the currently selected fill area interior style is PATTERN, the current pattern reference vectors attribute is used in conjunction with the current pattern width and height vectors to display the fill area and fill area set output primitives.

Most of the DEC GKS supported workstations do not fully support this function. They do accept the function call, but do not make any changes to the pattern.

SET PATTERN REPRESENTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GSPAR (WKID, PAI, DIMX, DIMY, ISC, ISR, DX, DY, COLIA)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
PAI	Integer	Read	Pattern bundle table index
DIMX, DIMY	Integer	Read	Dimensions of pattern array
ISC, ISR	Integer	Read	Indexes to the starting column and starting row in the pattern array
DX, DY	Integer	Read	Number of columns and rows used
COLIA (DIMX,DIMY)	Integer	Read	Pattern array

Description

The SET PATTERN REPRESENTATION function allows the user 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 the SET PATTERN REPRESENTATION function may cause DEC GKS to implicitly regenerate the workstation surface.

Attribute values passed to this function must be valid for the specified workstation. For more information concerning device-specific attributes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

CELL ARRAY
SET FILL AREA INTERIOR STYLE
SET FILL AREA STYLE INDEX
SET PATTERN REFERENCE POINT
SET PATTERN SIZE

SET PATTERN SIZE

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSPA (SZX, SZY)

Argument	Data Type	Access	Description
SZX, SZY	Real	Read	Pattern width and height in WC units

Description

The SET PATTERN SIZE function specifies the geometric attribute, *current pattern size* entry in the GKS state list, which is the height and width vectors in WC units.

DEC GKS begins replicating the pattern representation at the pattern reference point, and continues until the polygonal fill area in WC space is full. DEC GKS uses this value for all subsequent calls to FILL AREA until you specify another value.

Most of the DEC GKS supported workstations do not fully support this function. They do accept the function call, but do not make any changes to the pattern. For more information concerning patterns, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

FILL AREA
SET FILL AREA INTERIOR STYLE
SET FILL AREA STYLE INDEX

SET PICK IDENTIFIER

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSPKID (PKID)

Argument	Data Type	Access	Description
PKID	Integer	Read	New pick identifier

Description

The SET PICK IDENTIFIER function sets the *current pick identifier* entry in the GKS state list to the specified value. All subsequent output primitives stored in segments are assigned the value specified to the SET PICK IDENTIFIER function, 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.

See Also

GET PICK
REQUEST PICK
SAMPLE PICK
Example 9–2 for a program example using the SET PICK IDENTIFIER function

SET POLYLINE COLOUR INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSPLCI (COLI)

Argument	Data Type	Access	Description
COLI	Integer	Read	Polyline color index. The default value is 1, which designates the default foreground color.

Description

The SET POLYLINE COLOUR INDEX function sets the *current polyline color index* entry in the GKS state list to the specified index value. The specified index value is used for the display of subsequent polyline output primitives, created when the current polyline color index ASF is INDIVIDUAL. This value does not affect the display of subsequent polyline output primitives created when the current fill area color index ASF is BUNDLED.

If the specified color index is not present in a workstation color table, a workstation-dependent color index is used on that workstation.

See Also

SET COLOUR REPRESENTATION
Example 6–4 for a program example using a SET . . . COLOUR INDEX function

SET POLYLINE INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSPLI (PLI)

Argument	Data Type	Access	Description
PLI	Integer	Read	Polyline bundle table index. The default value is 1.

Description

The SET POLYLINE INDEX function establishes the index value pointing into the polyline bundle table.

The polyline bundle table contains entries for the attribute values, polyline color index, polyline type, and polyline width scale factor. When calling this function, DEC GKS uses the bundle table only if the corresponding ASF has been set to BUNDLED.

See Also

SET ASPECT SOURCE FLAGS
SET POLYLINE REPRESENTATION
Example 6–2 for a program example using a SET . . . INDEX function

SET POLYLINE REPRESENTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GSPLR (WKID, PLI, LTYPE, LWIDTH, COLI)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
PLI	Integer	Read	Polyline bundle table index
LTYPE	Integer (constant)	Read	Polyline type
LWIDTH	Real	Read	Polyline width scale factor
COLI	Integer	Read	Color index of polyline

Constants

Defined Argument	Constant	Description
LTYPE	GLSOL	Solid line. This is the default value.
	GLDASH	Dashed line.
	GLDOT	Dotted line.
	GLDASD	Dashed-dotted line.

Note

Other, nonstandard, polyline types are available. See Appendix B.

Description

The SET POLYLINE REPRESENTATION function allows the user 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 the SET POLYLINE REPRESENTATION function may cause DEC GKS to implicitly regenerate the workstation surface.

Attribute values passed to this function must be valid for the specified workstation. For more information concerning device-specific attributes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

SET ASPECT SOURCE FLAGS
SET LINETYPE
SET LINEWIDTH SCALE FACTOR
SET POLYLINE COLOUR INDEX
Example 6–1 for a program example using a SET . . . REPRESENTATION function

SET POLYMARKER COLOUR INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSPMCI (COLI)

Argument	Data Type	Access	Description
COLI	Integer	Read	Polymarker color index. The default value is 1, which designates the default foreground color.

Description

The SET POLYMARKER COLOUR INDEX function sets the *current polymarker color index* entry in the GKS state list to the specified value. The specified index value is used for the display of subsequent polymarker output primitives, created when the current polymarker color index ASF in the GKS state list is INDIVIDUAL. This value does not affect the display of subsequent polymarker output primitives created when the current fill area color index ASF in the GKS state list is BUNDLED.

If the specified color index is not present in a workstation color table, a workstation-dependent color index is used on that workstation.

See Also

SET ASPECT SOURCE FLAGS
SET POLYMARKER INDEX
SET POLYMARKER REPRESENTATION
Example 6–4 for a program example using the SET POLYMARKER COLOUR INDEX function

SET POLYMARKER INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSPMI (PMI)

Argument	Data Type	Access	Description
PMI	Integer	Read	Polymarker bundle table index. The default value is 1.

Description

The SET POLYMARKER INDEX function establishes the index value pointing into the polymarker bundle table. This table contains entries for the attribute values, polymarker color index, polymarker type, and polymarker size scale factor. When calling the SET POLYMARKER INDEX function, DEC GKS uses the bundle table only if the corresponding ASF has been set to BUNDLED.

See Also

SET ASPECT SOURCE FLAGS
SET POLYMARKER REPRESENTATION
Example 6–2 for a program example using a SET . . . INDEX function

SET POLYMARKER REPRESENTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GSPMR (WKID, PMI, MTYPE, MSZSF, COLI)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
PMI	Integer	Read	Polymarker bundle table index value
MTYPE	Integer (constant)	Read	Polymarker type
MSZSF	Real	Read	Polymarker size scale factor
COLI	Integer	Read	Polymarker color index

Constants

Defined Argument	Constant	Description
MTYPE	GPOINT	Dot.
	GPLUS	Plus sign.
	GAST	Asterisk. This is the default value.
	GOMARK	Circle.
	GXMARK	Diagonal cross.

Note

Other, nonstandard, polymarker types are available. See Appendix B.

Description

The SET POLYMARKER REPRESENTATION function allows the user to redefine an existing polymarker bundle table index representation, or to define a new polymarker bundle table index value, by specifying the polymarker type, the polymarker size, and the polymarker color index associated with the specified bundle index.

Depending on the capabilities of your workstation, a call to the SET POLYMARKER REPRESENTATION function may cause DEC GKS to implicitly regenerate the workstation surface.

Attribute values passed to this function must be valid for the specified workstation. For more information concerning device-specific attributes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

SET ASPECT SOURCE FLAGS
SET MARKER SIZE SCALE FACTOR
SET MARKER TYPE
SET POLYMARKER COLOUR INDEX
SET POLYMARKER INDEX
Example 6–1 for a program example using a SET . . . REPRESENTATION function

SET TEXT ALIGNMENT

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSTXAL (TXALH, TXALV)

Argument	Data Type	Access	Description
TXALH	Integer (constant)	Read	Horizontal alignment
TXALV	Integer (constant)	Read	Vertical alignment

Constants

Defined Argument	Constant	Description
TXALH	GAHNOR	Normal horizontal alignment. This is the default value.
	GALEFT	Left horizontal alignment.
	GACENT	Center horizontal alignment.
	GARITE	Right horizontal alignment.
TXALV	GAVNOR	Normal vertical alignment. This is the default value.
	GATOP	Top vertical alignment.
	GACAP	Cap vertical alignment.
	GAHALF	Half vertical alignment.
	GABASE	Base vertical alignment.
	GABOTT	Bottom vertical alignment.

Description

The SET TEXT ALIGNMENT function sets the *current text alignment* entry in the GKS state list to a value that specifies the positioning of the text extent rectangle.

DEC GKS uses the value specified in a call to SET TEXT ALIGNMENT for all subsequent calls to TEXT until you specify another value. Once you have determined the starting point, the text path (see the SET TEXT PATH function), and the character up vector (see the SET CHARACTER UP VECTOR function), you have in effect established an imaginary line running through the starting point, on which to output text. At this point, you can use this function to shift the text extent rectangle along this established line.

The values 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, but you can also position the rectangle vertically along the same line. For example, you can position the text extent rectangle so 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.

See Also

SET CHARACTER UP VECTOR
SET TEXT PATH
Example 6–4 for a program example using the SET TEXT ALIGNMENT function

SET TEXT COLOUR INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSTXCI (COLI)

Argument	Data Type	Access	Description
COLI	Integer	Read	Text color index. The default value is 1, which designates the default foreground color.

Description

The SET TEXT COLOUR INDEX function sets the *current text color index* entry in the GKS state list to the specified value.

If the current text font ASF is set to INDIVIDUAL, the text color index value is used in subsequent text and text 3 primitives. If the ASF setting is BUNDLED, the value has no effect. If the specified color is not available, a workstation-dependent color is used on that workstation.

See Also

SET COLOUR REPRESENTATION
Example 6–4 for a program example using a SET . . . COLOUR INDEX function

SET TEXT FONT AND PRECISION

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSTXFP (FONT, PREC)

Argument	Data Type	Access	Description
FONT	Integer	Read	Font value. The default value is 1.
PREC	Integer (constant)	Read	Precision value.

Constants

Defined Argument	Constant	Description
PREC	GSTRP	String precision. DEC GKS evaluates character height and width attributes only; this is the default precision value.
	GCHARP	Character precision. DEC GKS evaluates each character for compliance with all other specified text attributes.
	GSTRKP	Stroke precision. DEC GKS looks for exact compliance with all specified text attributes.

Description

The SET TEXT FONT AND PRECISION function sets the *current text font and precision* entry in the GKS state list to the specified value. In calls to this function, the types of fonts available depend on which precision value you pass as an argument. The values, in order of increasing precision, are as follows:

- String
- Character
- Stroke

As the precision value 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 a starting position for the string located outside of the current normalization viewport, a call to this function 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 specify character precision, a call to this function causes the text string to be clipped at the current normalization viewport on a character-by-character basis.

If you require string or character precision, 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, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

If you specify stroke precision, a call to this function 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.

Be aware that all images are clipped at the current workstation window.

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 number 1, and string precision.

See Also

SET TEXT REPRESENTATION

SET TEXT INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSTXI (TXI)

Argument	Data Type	Access	Description
TXI	Integer	Read	Text bundle index. The default value is 1.

Description

The SET TEXT INDEX function establishes the index value pointing into the text bundle table. This table contains entries for the attribute values, text font and precision, character expansion factor, character spacing, and text color index. When calling this function, DEC GKS uses the bundle table only if the corresponding ASF has been set to BUNDLED.

See Also

SET ASPECT SOURCE FLAGS
SET TEXT REPRESENTATION
Example 6–2 for a program example using a SET . . . INDEX function

SET TEXT PATH

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSTXP (TXP)

Argument	Data Type	Access	Description
TXP	Integer (constant)	Read	Text path

Constants

Defined Argument	Constant	Description
TXP	GRIGHT	Text string reads from left to right. This is the default value.
	GLEFT	Text string reads from right to left.
	GUP	Text string reads from bottom to top.
	GDOWN	Text string reads from top to bottom.

Description

The SET TEXT PATH function sets the geometric attribute, *current text path* entry in the GKS state list to be the writing direction for the display of text.

DEC GKS uses the value specified in a call to SET TEXT PATH for all subsequent calls to TEXT until you specify another value. Once you have determined the starting point and the character up vector (see the SET CHARACTER UP VECTOR function), 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 at the starting point (see the SET TEXT ALIGNMENT function). According to the current text path, the string reads either to the right along the imaginary line (the default), to the left along the imaginary line, upwards in a perpendicular direction from the imaginary line, or downwards in a perpendicular direction from the imaginary line.

If using the default text alignment (see the SET TEXT ALIGNMENT function), DEC GKS places the first letter of this string at the starting point, and subsequent letters are written along the imaginary line in the direction specified by a call to this function. The default text path is left to right along the imaginary line (text path RIGHT).

See Also

SET CHARACTER UP VECTOR
SET TEXT ALIGNMENT
Example 6–4 for a program example using the SET TEXT PATH function

SET TEXT REPRESENTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GSTXR (WKID, TXI, FONT, PREC, CHXP, CHSP, COLI)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
TXI	Integer	Read	Text index value
FONT	Integer	Read	Font value
PREC	Integer (constant)	Read	Text precision
CHXP	Real	Read	Character expansion factor
CHSP	Real	Read	Spacing value in WC
COLI	Integer	Read	Color index of text

Constants

Defined Argument	Constant	Description
PREC	GSTRP	String precision. DEC GKS evaluates character height and width attributes only; this is the default precision value.
	GCHARP	Character precision. DEC GKS evaluates each character for compliance with all other specified text attributes.
	GSTRKP	Stroke precision. DEC GKS looks for exact compliance with all specified text attributes.

Description

The SET TEXT REPRESENTATION function allows the user 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.

This function allows you to change numerous text attributes, including the character expansion factor and the character spacing. When you change the value for the character expansion factor, the new value is multiplied by the width-to-height ratio specified in the original font specification to determine the new character width. The character height remains the same.

If you change the character spacing, using a positive number increases the spacing between letters (for example, the value 0.1 sets spacing to 0.1 times the character height). Using a negative number decreases the spacing and characters may overlap. The value 0.0 makes the bodies of the characters adjacent, without any separating space other than that defined as part of the character body by the font design.

Depending on the capabilities of your workstation, a call to the SET TEXT REPRESENTATION function may cause DEC GKS to implicitly regenerate the workstation surface.

Attribute values passed to this function must be valid for the specified workstation. For more information concerning device-specific attributes, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

SET ASPECT SOURCE FLAGS
SET CHARACTER SPACING
SET TEXT FONT AND PRECISION
SET TEXT INDEX
Example 6–1 for a program example using a SET . . . REPRESENTATION function

6.6 Program Examples

Example 6–1 illustrates the use of the SET COLOUR REPRESENTATION function.

Example 6–1 SET COLOUR REPRESENTATION Function

```
C  This program calls the SET COLOUR REPRESENTATION function to change
C  the color representation of a particular index from COLOR1to COLOR2.
C  This program assumes an RGB color model. To avoid making such
C  an assumption, use the SET COLOUR MODEL function to set the color
C  model to RGB explicity.
C
C  NOTE: To keep the example concise, no error checking is performed.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER   COLOR1
      REAL      COL2_COMP1
      REAL      COL2_COMP2
      REAL      COL2_COMP3
      INTEGER   FIGURE
      INTEGER   DEVNUM
      INTEGER   ICL
      INTEGER   N_POINTS
      REAL      PX (20)
      REAL      PY (20)
      REAL      TIMEOUT
      INTEGER   WS_ID
      DATA      FIGURE     /1/
      DATA      TIMEOUT    /5.0/

C  Open the GKS and workstation environments.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Define the points and color1. Create a segment for the figure, set
C  the fill index to COLOR1, set the fill interior style to solid,
C  and draw a figure.

      COLOR1 = 4
      N_POINTS = 20

      PX(1) = 0.1
      PY(1) = 0.1
      PX(2) = 0.4
      PY(2) = 0.1
      PX(3) = 0.4
      PY(3) = 0.2
      PX(4) = 0.6
      PY(4) = 0.2
      PX(5) = 0.6
      PY(5) = 0.1
```

(continued on next page)

Example 6–1 (Cont.) SET COLOUR REPRESENTATION Function

```
      PX(6) = 0.9
      PY(6) = 0.1
      PX(7) = 0.9
      PY(7) = 0.4
      PX(8) = 0.8
      PY(8) = 0.4
      PX(9) = 0.8
      PY(9) = 0.6
      PX(10) = 0.9
      PY(10) = 0.6

      PX(11) = 0.9
      PY(11) = 0.9
      PX(12) = 0.6
      PY(12) = 0.9
      PX(13) = 0.6
      PY(13) = 0.8
      PX(14) = 0.4
      PY(14) = 0.8
      PX(15) = 0.4
      PY(15) = 0.9

      PX(16) = 0.1
      PY(16) = 0.9
      PX(17) = 0.1
      PY(17) = 0.6
      PX(18) = 0.2
      PY(18) = 0.6
      PX(19) = 0.2
      PY(19) = 0.4
      PX(20) = 0.1
      PY(20) = 0.4

      CALL GCRSG (FIGURE)
      CALL GSFACI (COLOR1)
      CALL GSFAIS (GSOLID)
      CALL GFA (N_POINTS, PX, PY)
      CALL GCLSG ()

C  Wait 5 seconds.

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIMEOUT, WS_ID, ICL, DEVNUM)


C  Change the color representation of COLOR1 to COLOR2. This will
C  change the fill color of the figure from COLOR1 to COLOR2.
C  Wait 5 5 seconds.

      COL2_COMP1 = 1.0
      COL2_COMP2 = 0.43
      COL2_COMP3 = 0.09

      CALL GSCR (WS_ID, COLOR1, COL2_COMP1,
     *COL2_COMP2, COL2_COMP3)
      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIMEOUT, WS_ID, ICL, DEVNUM)

C  Close the GKS and workstation environments.
```

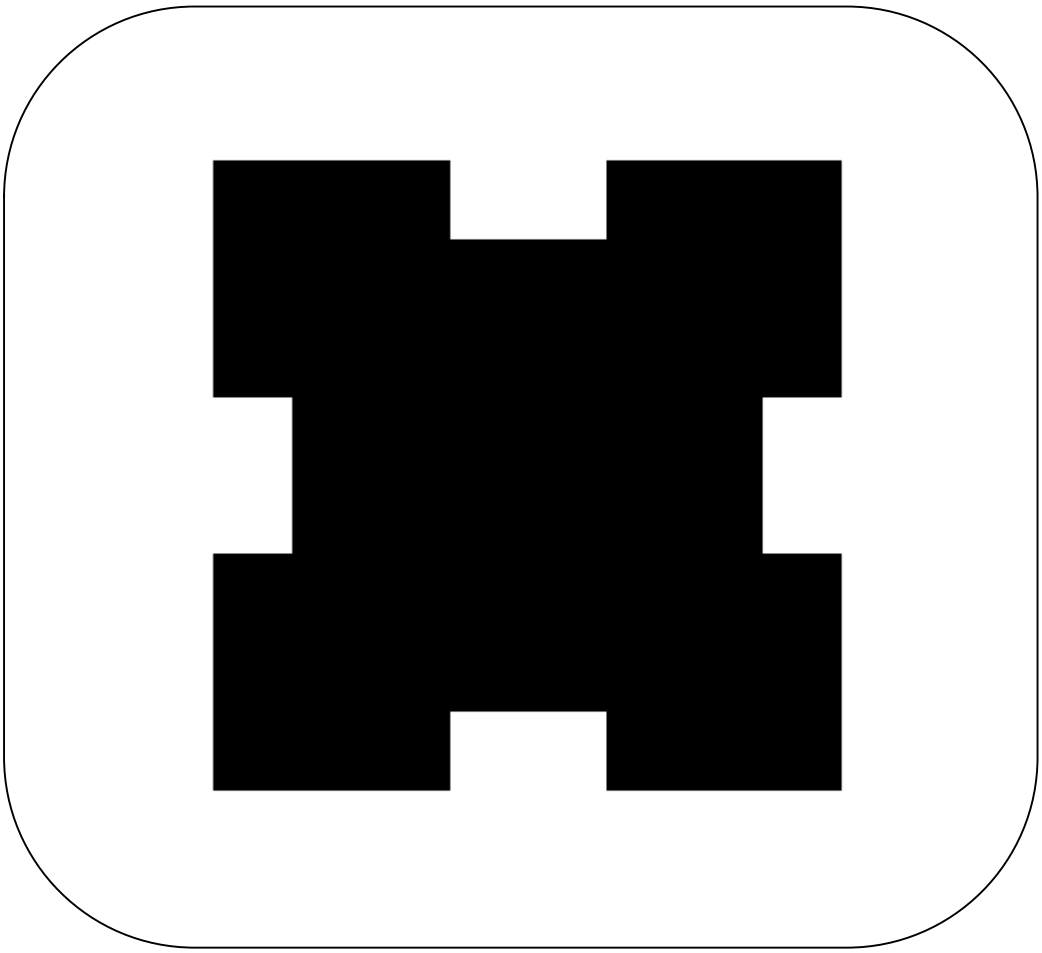
(continued on next page)

Example 6–1 (Cont.) SET COLOUR REPRESENTATION Function

```
      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 6–1 shows the program's effect on a VAXstation workstation running DECwindows software.

Figure 6–1 SET COLOUR REPRESENTATION Output



ZK–4010A–GE

Example 6–2 illustrates the use of the SET FILL AREA REPRESENTATION function.

Example 6–2 SET FILL AREA REPRESENTATION Function

```
C  This program sets the attribute source flags (ASFs) to BUNDLED,
C  shows the fill area corresponding to the index 6, then changes
C  the attributes associated with fill area index 6, using the
C  SET FILL AREA REPRESENTATION function.
C
C  NOTE: To keep the example concise, no error checking is performed.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER DEVNUM
      INTEGER FIGURE
      INTEGER FILL_INDEX
      INTEGER FLAGS (13)
      INTEGER ICL
      INTEGER INCR
      INTEGER NEW_COLOR
      INTEGER NEW_STYLE
      INTEGER N_FLAGS
      INTEGER N_POINTS
      REAL PX (20)
      REAL PY (20)
      REAL TIMEOUT
      INTEGER WS_ID
      DATA FLAGS    /13*0/
      DATA TIMEOUT  /5.0/
      DATA FIGURE   /1/

C  Open the GKS and workstation environments.

      WS_ID = 1

      CALL GOPKS (6,0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Set the attribute source flags (ASFs) to BUNDLED.

      INCR = 1
      N_FLAGS = 1
      FLAGS (INCR) = GBUNDL

      CALL GSASF (FLAGS)


C  Define the points and show the fill area corresponding to an index value
C  of 6.

      FILL_INDEX = 6
      N_POINTS = 20
      PX(1) = 0.1
      PY(1) = 0.1
      PX(2) = 0.4
      PY(2) = 0.1
      PX(3) = 0.4
      PY(3) = 0.2
      PX(4) = 0.6
      PY(4) = 0.2
      PX(5) = 0.6
      PY(5) = 0.1
```

(continued on next page)

Example 6–2 (Cont.) SET FILL AREA REPRESENTATION Function

```
      PX(6) = 0.9
      PY(6) = 0.1
      PX(7) = 0.9
      PY(7) = 0.4
      PX(8) = 0.8
      PY(8) = 0.4
      PX(9) = 0.8
      PY(9) = 0.6
      PX(10) = 0.9
      PY(10) = 0.6

      PX(11) = 0.9
      PY(11) = 0.9
      PX(12) = 0.6
      PY(12) = 0.9
      PX(13) = 0.6
      PY(13) = 0.8
      PX(14) = 0.4
      PY(14) = 0.8
      PX(15) = 0.4
      PY(15) = 0.9

      PX(16) = 0.1
      PY(16) = 0.9
      PX(17) = 0.1
      PY(17) = 0.6
      PX(18) = 0.2
      PY(18) = 0.6
      PX(19) = 0.2
      PY(19) = 0.4
      PX(20) = 0.1
      PY(20) = 0.4

C  Put all output in a segment, and show the fill area corresponding to the
C  index value 6.

      FIGURE = 1

      CALL GCRSG (FIGURE)
      CALL GSFAI (FILL_INDEX)
      CALL GFA (N_POINTS, PX, PY)
      CALL GCLSG ()

C  Wait 5 seconds.

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIMEOUT, WS_ID, ICL, DEVNUM)

C  Change the attributes associated with a fill area index value of 6.

      NEW_COLOR = 1
      NEW_STYLE = -9

      CALL GSFAR (WS_ID, FILL_INDEX, GHATCH,
     *NEW_STYLE, NEW_COLOR)

C  Cause a regeneration of the screen to see the change on a workstation.

      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIMEOUT, WS_ID, ICL, DEVNUM)

C  Close the GKS and workstation environments.
```

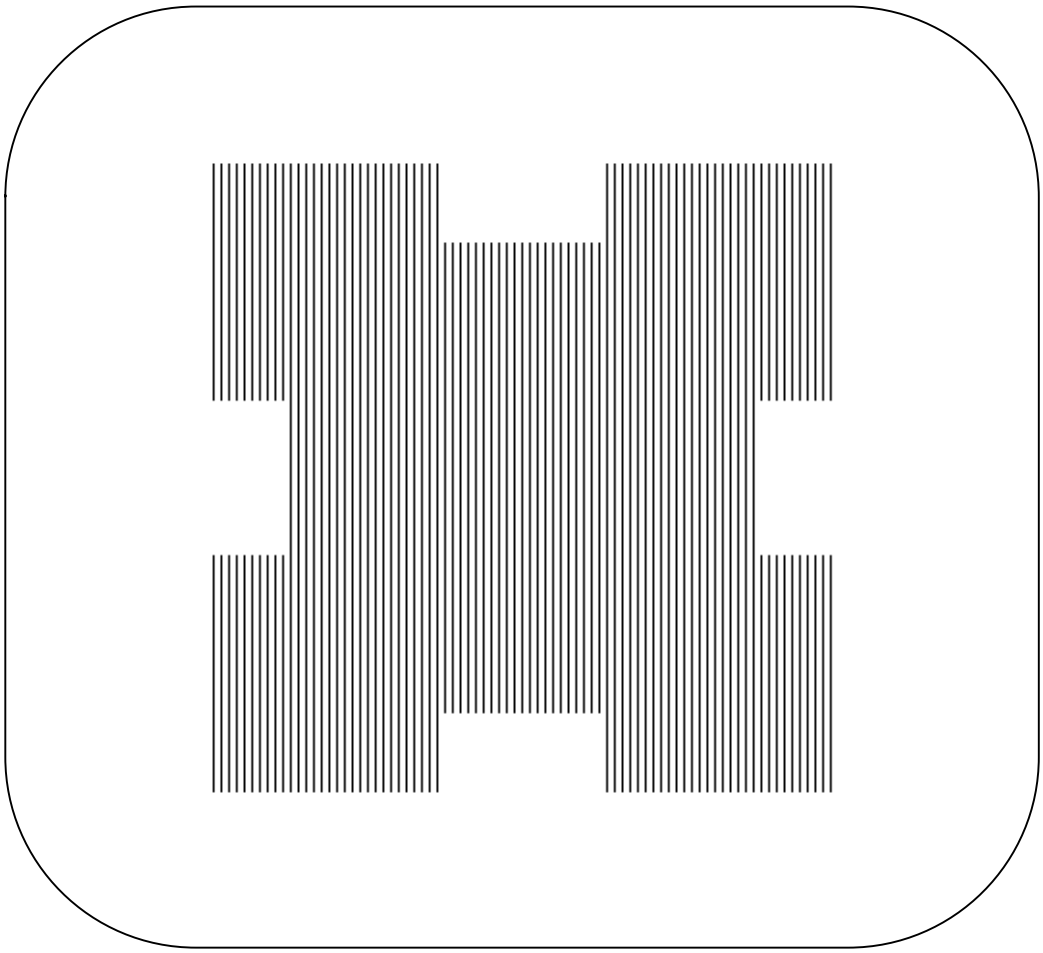
(continued on next page)

Example 6–2 (Cont.) SET FILL AREA REPRESENTATION Function

```
CALL GDAWK (WS_ID)
CALL GCLWK (WS_ID)
CALL GCLKS ()
END
```

Figure 6–2 shows the program's effect on a VAXstation workstation running DECwindows software.

Figure 6–2 SET FILL AREA REPRESENTATION Output



ZK–4011A–GE

Example 6–3 illustrates the use of the SET LINETYPE function.

Example 6–3 SET LINETYPE Function

```
C  This program calls the SET LINETYPE function to set the
C  line type to the dashed and dotted line. The program
C  draws a line figure displaying the set line type.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER DEVNUM
      INTEGER ICL
      INTEGER N_POINTS
      REAL PX (29)
      REAL PY (29)
      REAL TIMEOUT
      INTEGER WS_ID
      DATA TIMEOUT    /5.0/

C  Open the GKS and workstation environments.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Define the line points, set the line type to dashed and dotted lines, and
C  draw the line figure.

      N_POINTS = 29
      PX(1) = 0.4
      PY(1) = 0.9
      PX(2) = 0.1
      PY(2) = 0.9
      PX(3) = 0.1
      PY(3) = 0.6
      PX(4) = 0.2
      PY(4) = 0.6
      PX(5) = 0.2
      PY(5) = 0.4

      PX(6) = 0.1
      PY(6) = 0.4
      PX(7) = 0.1
      PY(7) = 0.1
      PX(8) = 0.4
      PY(8) = 0.1
      PX(9) = 0.4
      PY(9) = 0.2
      PX(10) = 0.6
      PY(10) = 0.2

      PX(11) = 0.6
      PY(11) = 0.1
      PX(12) = 0.9
      PY(12) = 0.1
      PX(13) = 0.9
      PY(13) = 0.4
      PX(14) = 0.8
      PY(14) = 0.4
      PX(15) = 0.8
      PY(15) = 0.6
```

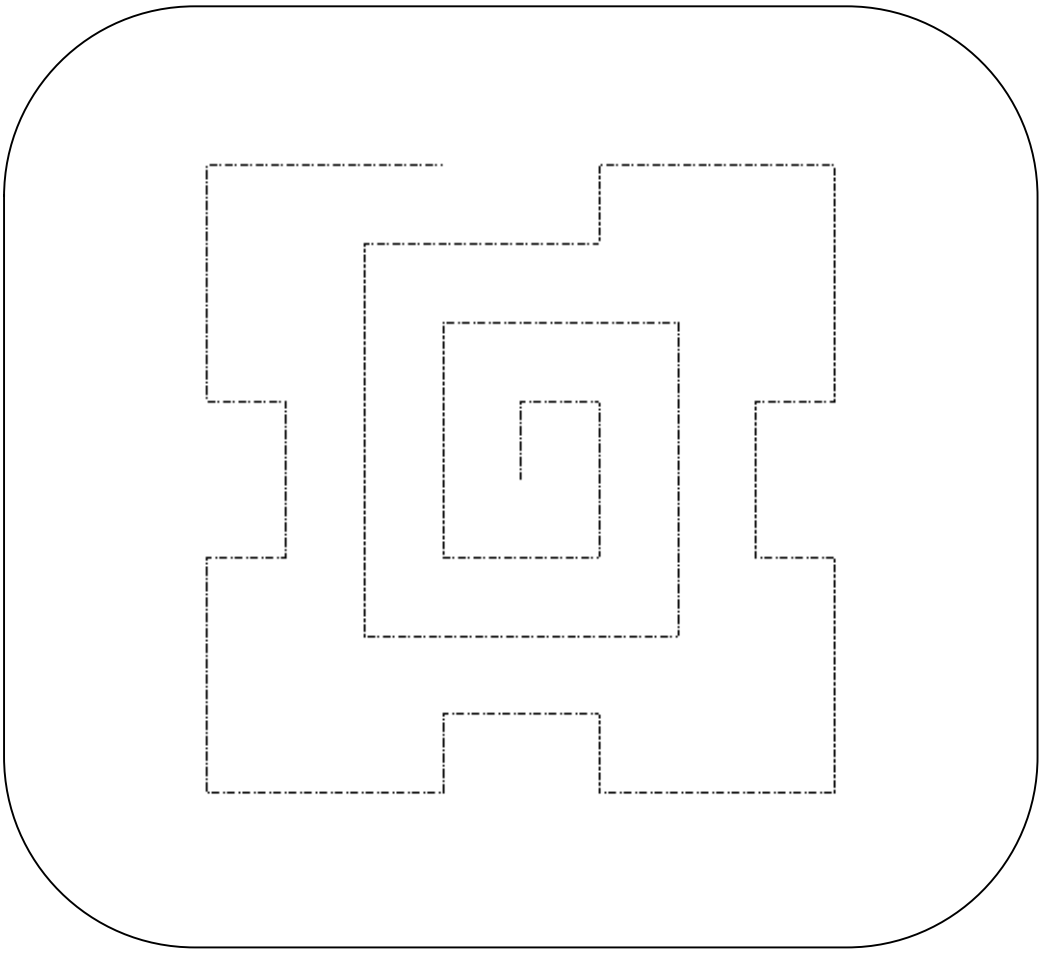
(continued on next page)

Example 6–3 (Cont.) SET LINETYPE Function

```
      PX(16) = 0.9
      PY(16) = 0.6
      PX(17) = 0.9
      PY(17) = 0.9
      PX(18) = 0.6
      PY(18) = 0.9
      PX(19) = 0.6
      PY(19) = 0.8
      PX(20) = 0.3
      PY(20) = 0.8

      PX(21) = 0.3
      PY(21) = 0.3
      PX(22) = 0.7
      PY(22) = 0.3
      PX(23) = 0.7
      PY(23) = 0.7
      PX(24) = 0.4
      PY(24) = 0.7
      PX(25) = 0.4
      PY(25) = 0.4

      PX(26) = 0.6
      PY(26) = 0.4
      PX(27) = 0.6
      PY(27) = 0.6
      PX(28) = 0.5
      PY(28) = 0.6
      PX(29) = 0.5
      PY(29) = 0.5

      CALL GSLN (GLDASD)
      CALL GPL (N_POINTS, PX, PY)

C  Wait 5 seconds.

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIMEOUT, WS_ID, ICL, DEVNUM)

C  Close the GKS and workstation environments.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 6–3 shows the program's effect on a VAXstation workstation running DECwindows software.

Figure 6–3 SET LINETYPE Output



ZK–4012A–GE

Example 6–4 illustrates the use of the SET TEXT ALIGNMENT function.

Example 6–4 SET TEXT ALIGNMENT Function

```
C  This program calls the SET TEXT ALIGNMENT function to write
C  a string to the workstation using the normal text alignments
C  for each of four text paths.

      IMPLICIT NONE
      INCLUDE 'gks.f'
```

(continued on next page)

Example 6–4 (Cont.) SET TEXT ALIGNMENT Function

```
      INTEGER DEVNUM
      INTEGER ICL
      REAL LARGER
      INTEGER ONE_PMARK
      INTEGER RED
      REAL START_PT_X
      REAL START_PT_Y
      REAL TIMEOUT1
      REAL TIMEOUT2
      INTEGER WS_ID
      DATA ONE_PMARK  /1/
      DATA TIMEOUT1   /1.00/
      DATA TIMEOUT2   /5.00/
      DATA RED        /2/

C  Open the GKS and workstation environments.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Initialize the starting point for each of the text lines.

      START_PT_X = 0.5
      START_PT_Y = 0.5

C  Set the polymarker color index, and the polymarker type.
C  Draw the polymarker to provide a point of reference for
C  the lines of text.

      CALL GSPMCI (RED)
      CALL GSMK (GPLUS)
      CALL GPM (ONE_PMARK, START_PT_X, START_PT_Y)
      CALL GUWK (WS_ID, GPOSTP)

C  Set the text character height and the text alignment.

      LARGER = 0.07

      CALL GSCHH (LARGER)
      CALL GSTXAL (GAHNOR, GAVNOR)


C  Set a rightward text path and write a character string.
C  Wait 1 second.

      CALL GSTXP (GRIGHT)
      CALL GTX (START_PT_X, START_PT_Y, ' TEXT LINE 1')
      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIMEOUT1, WS_ID, ICL, DEVNUM)

C  Set a leftward text path and write a character string.
C  Wait 1 second.

      CALL GSTXP (GLEFT)
      CALL GTX (START_PT_X, START_PT_Y, ' TEXT LINE 2')
      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIMEOUT1, WS_ID, ICL, DEVNUM)
```

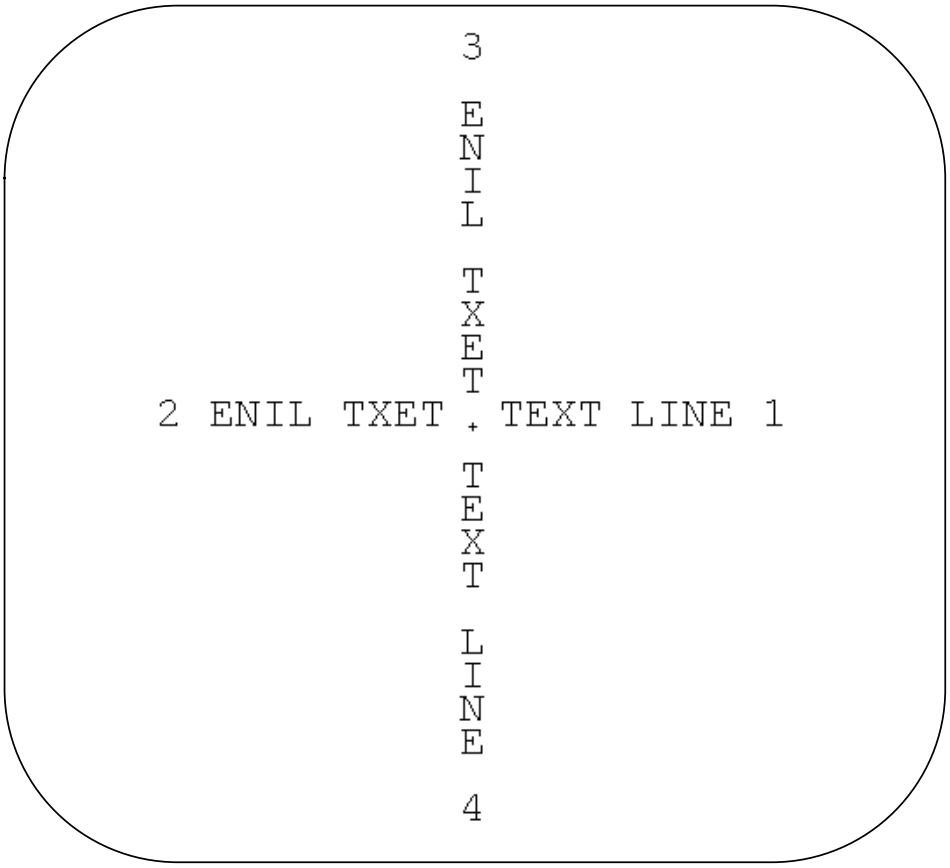
(continued on next page)

Example 6–4 (Cont.) SET TEXT ALIGNMENT Function

```
C  Set an upward text path and write a character string.
C  Wait 1 second.

      CALL GSTXP (GUP)
      CALL GTX (START_PT_X, START_PT_Y, ' TEXT LINE 3')
      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIMEOUT1, WS_ID, ICL, DEVNUM)

C  Set a downward text path and write a character string.
C  Wait 5 seconds.

      CALL GSTXP (GDOWN)
      CALL GTX (START_PT_X, START_PT_Y, ' TEXT LINE 4')
      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIMEOUT2, WS_ID, ICL, DEVNUM)

C  Close the GKS and workstation environments.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 6–4 shows the program's effect on a VAXstation workstation running DECwindows software.

Figure 6–4 SET TEXT ALIGNMENT Output



ZK–4009A–GE

Transformation Functions

Insert tabbed divider here. Then discard this sheet.

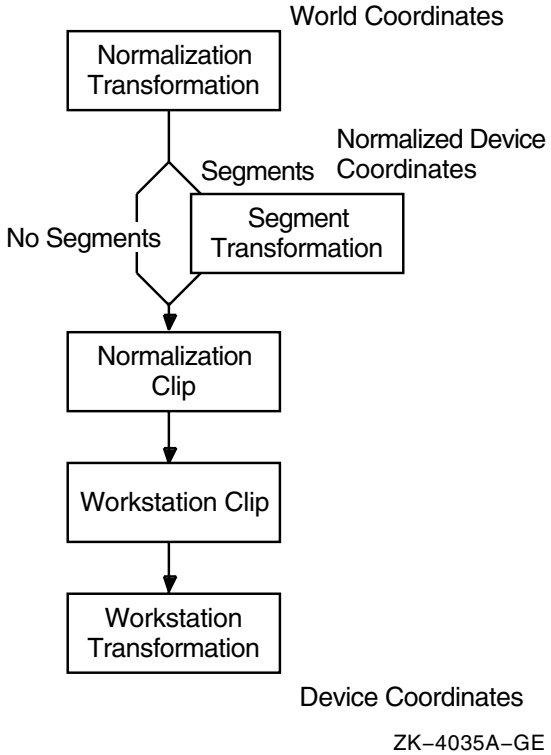
7
Transformation Functions

The DEC GKS transformation functions allow you to compose a picture, control how much of the picture is displayed on the workstation surface, and control how much of the workstation surface is used to display the picture.

When you request input and generate output on the workstation surface, you actually work with a number of coordinate systems. The image is transformed from one coordinate system to the next.

Using DEC GKS, you work with a transformation pipeline. The transformation pipeline consists of a number of transformations that affect various coordinate systems. To meet the needs of graphics programming, DEC GKS supports both the two-dimensional and three-dimensional transformation pipelines. Figure 7–1 illustrates the two-dimensional pipeline.

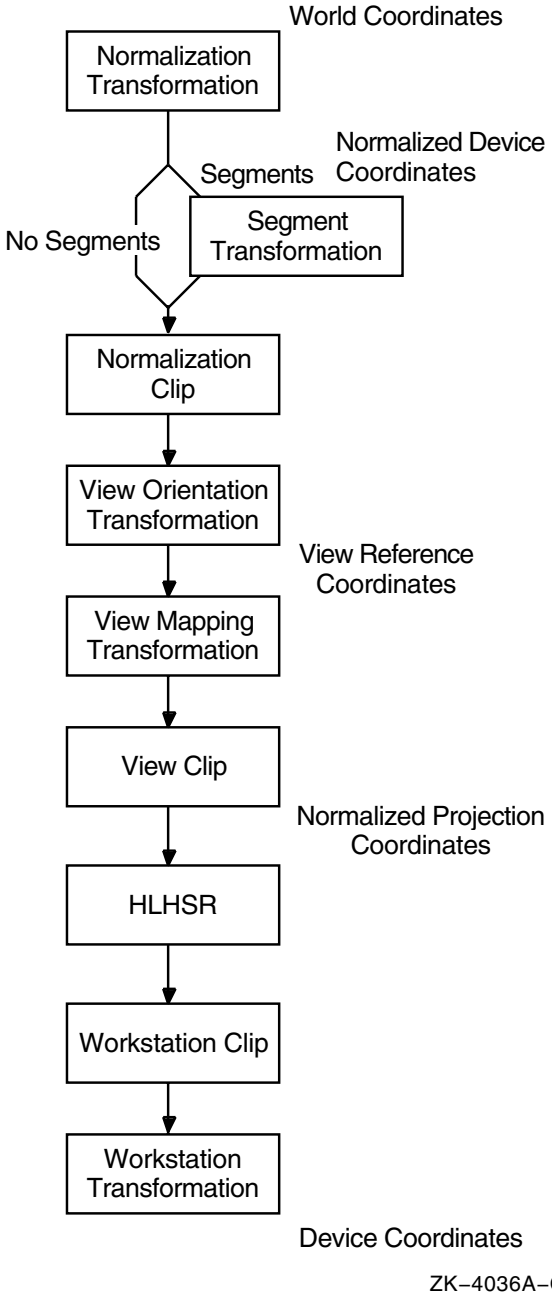
Figure 7–1 The DEC GKS Two-Dimensional Transformation Pipeline



ZK–4035A–GE

The three coordinate systems work as a pipeline and ultimately create a two-dimensional object on your physical display device. You use portions of the world coordinate (WC) system to plot the output primitives, a portion of the device-independent normalized device coordinate (NDC) plane to compose a complete picture, and a portion of the device coordinate plane to present all or part of your picture on all or part of the surface of the workstation. Figure 7–2 illustrates the three-dimensional pipeline.

Figure 7–2 The DEC GKS Three-Dimensional Transformation Pipeline



The five transformations illustrated in Figure 7–2 work as a pipeline and ultimately create a three-dimensional object on your physical display device. You use portions of the WC system to plot the output primitives, a portion of the device-independent NDC plane to compose a complete picture, portions of the view reference coordinate (VRC) system to orient the picture, portions of the normalized projection coordinate (NPC) system to determine projection volume, and a portion of the device coordinate plane to present all or part of your picture on all or part of the surface of the workstation.

For both transformation pipelines, the WC system is an imaginary coordinate plane used to plot a graphic image. The NDC system is a device-independent, imaginary coordinate plane on which you compose a picture using designated portions of the WC plane. Once you compose a two-dimensional picture in the NDC space, you can display all or part of the picture in NDC space on the surface of the physical device. If the picture is three-dimensional, you can orient (translate and rotate) your picture through the VRC system. The VRC image is then projected to the NPC system. The NPC system lets you determine how much of the picture plotted in WC points will be mapped to the device coordinate system. You can display all or part of the picture in NPC space on the surface of the physical device.

When you call one of the DEC GKS output functions, you specify WC points. Using a series of default windows and viewports, the output primitive is transformed from an image on the WC plane to an image on the NDC plane, oriented and clipped through the VRC and NPC systems if the picture is three-dimensional, and is finally transformed 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 WC 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 WC system is an imaginary, Cartesian coordinate system whose X and Y axes extend infinitely in all four directions. If you are using the three-dimensional pipeline, the X, Y, and Z axes extend infinitely in all six directions. The origin of the two-dimensional system is the point (0.0, 0.0). The origin of the three-dimensional system is (0.0, 0.0, 0.0). Depending on the type of data needed to plot your images, you can use any portion of the WC plane. For example, if the necessary data contains negative numbers, you can use the portions of the WC system that extend into the negative portions of the axes.

By default, DEC GKS, for two dimensions, transforms images according to a planar WC 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 the X and Y axes. For three dimensions, DEC GKS, transforms images according to a volumetric WC range whose lower left corner is the point (0.0, 0.0, 0.0) and whose sides extend from the point 0.0 to 1.0 on the X, Y, and Z axes. The 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 window, those points may or may not be part of the image on the NDC plane

depending on the current **clipping** indicator. Clipping is described in detail in Section 7.1.1. Example 7–3 illustrates resetting the normalization viewport.

7.1.1 The Normalized Device Coordinate System

As mentioned in the previous section, the normalization transformation is the transposition of WC points to NDC points. The NDC system is a device-independent coordinate plane on which you compose graphic pictures. The two-dimensional NDC system has X and Y axes that, in theory, extend 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. The three-dimensional NDC system has X, Y, and Z axes that, in theory, extend infinitely in all six directions with an origin at point (0.0, 0.0, 0.0); but in practice, only images contained in the range ([0,1] x [0,1] x [0,1]) can ultimately be transformed to the surface of a physical device.

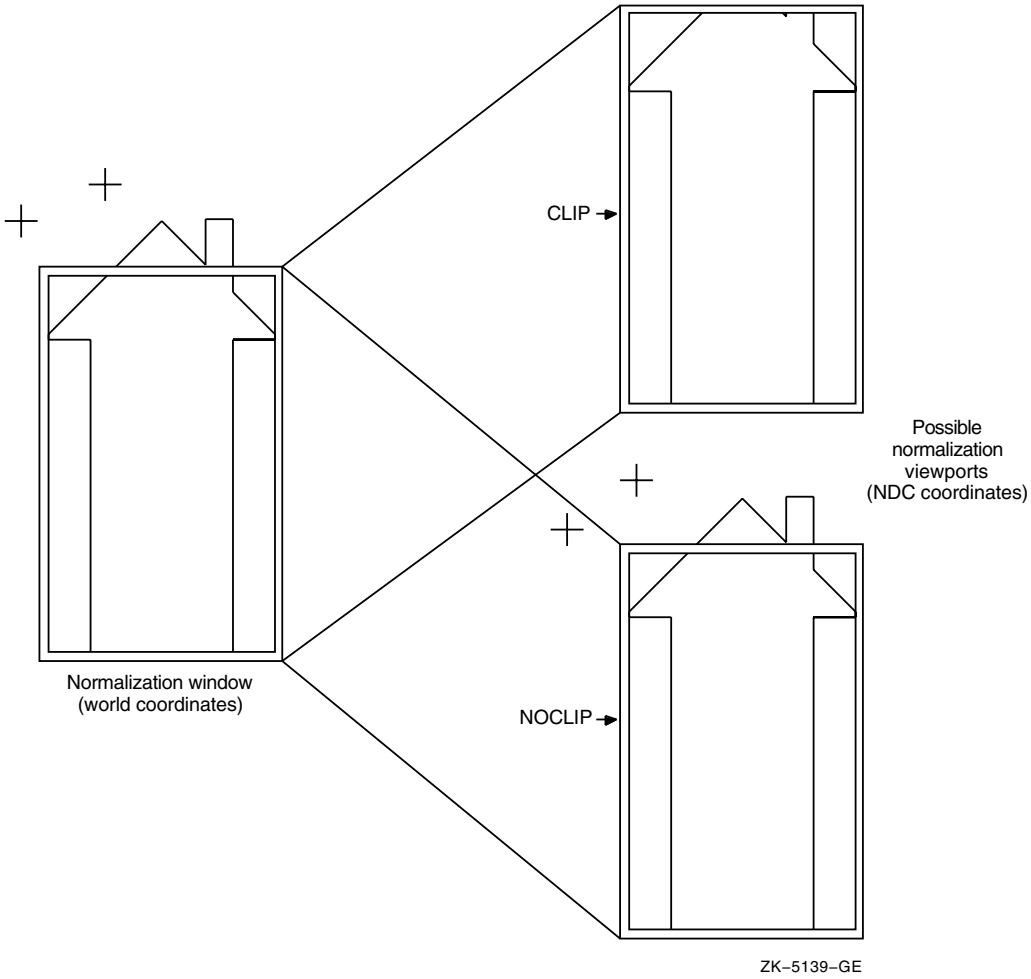
When DEC GKS transforms an image from the normalization window to the NDC plane, there must be a corresponding volume on which to map the contents of the window. This volume portion of the NDC space is called the normalization **viewport**. The two-dimensional default viewport has the range ([0,1] x [0,1]), and the three-dimensional default viewport has the range ([0,1] x [0,1] x [0,1]), in NDC points.

By default, DEC GKS maps the normalization window ([0,1] x [0,1]) in WC points to the viewport ([0,1] x [0,1]) in NDC points. 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. DEC GKS for three dimensions maps the normalization window ([0,1] x [0,1] x [0,1]) in WC points to the viewport ([0,1] x [0,1] x [0,1]) in NDC points.

Think of the normalization process as a way of transposing a number of areas of the WC plane onto the NDC plane with respect to the current normalization window and viewport. For example, DEC GKS maps the contents of the current normalization window onto the current viewport. If clipping is enabled (which is the default), the effect is like cutting the window from the WC 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 also maps the points that lie outside of the normalization window boundary to NDC space outside of the normalization viewport, but within the two-dimensional range ([0,1] x [0,1]), or the three-dimensional range ([0,1] x [0,1] x [0,1]).

Because DEC GKS clips images at the boundary of the normalization viewport, this viewport is also called the **clipping volume**. You can enable and disable clipping by calling the function SET CLIPPING INDICATOR. Figure 7–3 illustrates the clipping process according to the argument passed to SET CLIPPING INDICATOR.

Figure 7–3 The Clipping Rectangle



When creating a picture, consider that you can select different normalization transformations with different windows and viewports, thus mapping various portions of the WC space onto different portions of the NDC space. (In DEC GKS, valid normalization transformation numbers range from 0 to 255, and you 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 WC space as a scratch pad and the NDC space as a pasteboard on which to compose an entire picture. For example, if you want an output primitive to appear on the right side of a picture displayed 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 in the NDC space , you can output all or part of the picture to all or part of various workstation surfaces. By selecting a different normalization transformation with a different viewport, you can transpose the same window onto another portion of the NDC space.

7.1.2 Overlapping Viewports

When you define normalization viewports, it is possible to cause them to overlap in NDC space. 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 on segments, see Chapter 8.)

To illustrate the need for a viewport priority list during input, consider two viewports: the viewport of the unity (identity) transformation number 0 having the two-dimensional range ([0,1] x [0,1]), or the three-dimensional range ([0,1] x [0,1] x [0,1]), and a viewport, belonging to normalization transformation number 1, having the two-dimensional range ([0.5,1] x [0.5,1]), or the three-dimensional range ([0.5,1] x [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, which returns one point (locator) or a series of points (stroke) in device coordinates. DEC GKS translates the device coordinate points to NDC points. (Section 7.3 describes this process in detail.)

Once the device coordiante points are transformed to NDC points, DEC GKS must transform the NDC points to WC points. To transform the point, DEC GKS transforms the point from its viewport (NDC) value to the corresponding window (WC) value. However, if the user chooses a point on the right half of the default viewport, DEC GKS must calculate whether to use the unity viewport or the overlapping viewport of transformation number 1 to transform the point to WC values. DEC GKS needs to know to which normalization window the point is to be mapped: the window that corresponds to either normalization transformation number 0 or number 1.

To calculate 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 on).

To change the order of the viewport input priority list, call the function SET VIEWPORT INPUT 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.

If you call SET VIEWPORT INPUT PRIORITY to give transformation number 1 a higher transformation (1 higher than 0, 0 higher than 2, 2 higher than 3, and so on), 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, see Chapter 9.

7.2 View Transformations

View transformations apply only to the three-dimensional transformation pipeline.

Once your object is defined in NDC space you need to tell DEC GKS from which direction you are looking at your picture and what direction is up. The **viewing transformation** is the mechanism that lets you accomplish this.

The viewing transformation is workstation-dependent because it requires the use of information stored in the state list or description table of the output workstations.

Each workstation stores a workstation-specific number of view entries in a **view table**, which is part of the workstation state list. The view entries are numbered consecutively, starting with 0. View 0 is predefined to the identity transformation and cannot be modified. Other entries can be modified with the function SET VIEW REPRESENTATION 3. Each view entry contains a view orientation matrix, a view mapping matrix and clipping information.

You can change the orientation (translate and rotate) your picture by employing the **view orientation matrix** to define what direction you are viewing the picture from, as well as what direction is up. DEC GKS computes the view orientation matrix using the EVALUATE VIEW ORIENTATION MATRIX 3 function. The view orientation matrix establishes the VRC system (the UVN axes). The view orientation matrix is used to map each NDC point to an appropriate point in the VRC system. Although the VRC points are in the same units as NDC points, the VRC system is effectively a shifted and rotated version of the NDC system.

Once your picture is oriented in the VRC system, it is mapped to the NPC system. DEC GKS computes the **view mapping matrix** using the EVALUATE VIEW MAPPING MATRIX 3 function. The view transformation employs the view mapping matrix to map the VRC points to NPC points.

This transformation allows you to select a parallel or perspective projection for your picture. For more information on viewing, and parallel and perspective projections, see the *DEC GKS User's Guide*.

Each workstation can select, or clip, some part of its NPC space to be displayed somewhere on the physical display device of the workstation. You can clip your picture in NPC space according to the defined view clipping limits. The six NPC points are set by the SET VIEWPORT 3 function. For more information on clipping, see *DEC GKS User's Guide*.

7.3 Device Transformations

DEC GKS must map the picture on the two-dimensional NDC plane, or the three-dimensional NPC space, to the surface of one or more workstations. To do this, DEC GKS uses a second window and viewport called the **workstation window** and the **workstation viewport**. The workstation window is the rectangular portion of the two-dimensional NDC plane, or the rectangular parallelepiped of the three-dimensional NPC space, that is mapped to the workstation viewport. The workstation viewport is a portion of the display space. There can be numerous normalization transformations, but only one current workstation window and one current workstation viewport.

DEC GKS uses a default workstation window of the two-dimensional range ([0,1] x [0,1]), or the three-dimensional range ([0,1] x [0,1] x [0,1]). 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]) for two dimensions, or ([0,1] x [0,1] x [0,1]) for three dimensions. DEC GKS clips all points that exceed the default workstation window boundaries before it transforms the picture to device coordinate points, regardless of the current clipping flag setting.

If you are using the two-dimensional pipeline, the normalization transformation composes the picture in NDC space, and the workstation transformation presents all or part of the picture on all or part of the device surface. For example, by setting the workstation window with the SET WORKSTATION WINDOW function, 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's Guide* describes this process in detail.

If you are using the three-dimensional pipeline, the view transformation composes the picture in NPC space, and the workstation transformation presents all or part of the picture on all or part of the device surface. For example, by setting the workstation window with the SET WORKSTATION WINDOW 3 function, 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's Guide* describes 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 for two dimensions, or dissimilar maximum X, Y, and Z coordinate values for three dimensions.

To determine the maximum boundary of the workstation viewport, you should use the function, INQUIRE DISPLAY SPACE SIZE (3), which returns the maximum X and Y values (X, Y, and Z values for three dimensions) of the workstation display surface. (For more information, see Chapter 11, and SET WORKSTATION VIEWPORT (3) in this chapter.)

When you set the workstation window (by calling SET WORKSTATION WINDOW (3)) or the workstation viewport (by calling SET WORKSTATION VIEWPORT (3)), 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. Example 7–4 illustrates how to change the workstation window and viewport on a device that suppresses implicit regenerations. The *DEC GKS User's Guide* contains examples of working with the proportions of workstation windows and viewports.

7.4 Transformation Inquiries

You can use the following inquiry functions to obtain transformation information when writing device-independent code:

INQUIRE CLIPPING (3)
INQUIRE CURRENT NORMALIZATION TRANSFORMATION NUMBER
INQUIRE DISPLAY SPACE SIZE (3)
INQUIRE LIST OF NORMALIZATION TRANSFORMATION NUMBERS
INQUIRE MAXIMUM NORMALIZATION TRANSFORMATION
INQUIRE NORMALIZATION TRANSFORMATION (3)
INQUIRE WORKSTATION TRANSFORMATION (3)

For more information concerning device-independent programming, see the *DEC GKS User's Guide*. For more information on the inquiry functions, see Chapter 11.

7.5 Function Descriptions

This section describes the DEC GKS transformation functions in detail.

ACCUMULATE TRANSFORMATION MATRIX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GACTM (MI, X0, Y0, DX, DY, PHI, SCX, SCY, SW, MO)

Argument	Data Type	Access	Description
MI(2,3)	Array of reals	Read	[2×3] input transformation matrix, created previously by a call to either EVALUATE TRANSFORMATION MATRIX or ACCUMULATE TRANSFORMATION MATRIX: $\begin{bmatrix} MI(1,1)\,MI(1,2)\,MI(1,3) \\ MI(2,1)\,MI(2,2)\,MI(2,3) \end{bmatrix}$
X0, Y0	Real	Read	Fixed point for rotation These values can be expressed as WC or NDC values, depending on the value passed to the *SW* argument.
DX, DY	Real	Read	Shift vector. These values can be expressed as WC or NDC values, depending on the value passed to the *SW* argument. To avoid translation, pass the value 0.0 for these arguments.
PHI	Real	Read	Rotation angle, in radians (360 degrees = 2*pi). To avoid rotation, pass the value 0.0 for the argument.
SCX, SCY	Real	Read	Scale factors. To avoid scaling, pass the value 1.0 for these arguments.
SW	Integer (constant)	Read	Flag for WC or NDC values.

Argument	Data Type	Access	Description
MO(2,3)	Array of reals	Write	Output transformation matrix that results from evaluating the scaling, rotation, and shifting component values. Use this value as an argument to the SET SEGMENT TRANSFORMATION or INSERT SEGMENT function to establish a segment transformation. Declare this argument as a 6-element array, as follows: $\begin{bmatrix} MO(1,1)\,MO(1,2)\,MO(1,3) \\ MO(2,1)\,MO(2,2)\,MO(2,3) \end{bmatrix}$

Constants

Defined Argument	Constant	Description
SW	GWC	The fixed point and shift vectors are WC values.
	GNDC	The fixed point and shift vectors are NDC values.

Note

Use caution when specifying GWC. DEC GKS uses the *current* normalization transformation to transform the fixed point from WC 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.

Description

The ACCUMULATE TRANSFORMATION MATRIX function 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 order of transformation is:

1. Specified input matrix
2. Scale (relative to the specified fixed point)
3. Rotate (relative to the specified fixed point)
4. Shift

See the *DEC GKS User's Guide* for a description of segment transformation and transformation matrixes.

See Also

EVALUATE TRANSFORMATION MATRIX
INSERT SEGMENT
SET SEGMENT TRANSFORMATION
Example 7–1 for a program example using the ACCUMULATE TRANSFORMATION MATRIX function

ACCUMULATE TRANSFORMATION MATRIX 3

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GACTM3 (MI, X0, Y0, Z0, DX, DY, DZ, ROTX, ROTY, ROTZ, SCX, SCY, SCZ, SW, MO)

Argument	Data Type	Access	Description
MI(3,4)	Array of reals	Read	[3×4] input transformation matrix, created previously by a call to either EVALUATE TRANSFORMATION MATRIX 3 or ACCUMULATE TRANSFORMATION MATRIX 3: $\begin{bmatrix} MI(1,1) & MI(1,2) & MI(1,3) & MI(1,4) \\ MI(2,1) & MI(2,2) & MI(2,3) & MI(2,4) \\ MI(3,1) & MI(3,2) & MI(3,3) & MI(3,4) \end{bmatrix}$
X0, Y0, Z0	Real	Read	Fixed point for rotation.
DX, DY, DZ	Real	Read	Shift vector. To avoid translation, pass the value 0 for all three arguments.
ROTX, ROTY, ROTZ	Real	Read	Rotation angle, in radians (360 degrees = 2*pi). To avoid rotation, pass the value 0 for these arguments.
SCX, SCY, SCZ	Real	Read	Scale factors. To avoid scaling, pass the value 1.0 for these arguments.
SW	Integer (constant)	Read	Flag for WC or NDC values.
MO(3,4)	Array of reals	Write	Output transformation matrix that results from evaluating the scaling, rotation, and shifting component values. Use this value as an argument to the SET SEGMENT TRANSFORMATION 3 or INSERT SEGMENT 3 function to establish a segment transformation. Declare this argument as a 12-element array, as follows:

Argument	Data Type	Access	Description
			$\begin{bmatrix} MO(1,1) MO(1,2) MO(1,3) MO(1,4) \\ MO(2,1) MO(2,2) MO(2,3) MO(2,4) \\ MO(3,1) MO(3,2) MO(3,3) MO(3,4) \end{bmatrix}$

Constants

Defined Argument	Constant	Description
SW	GWC	The fixed point and shift vectors are WC values.
	GNDC	The fixed point and shift vectors are NDC values.

Note

Use caution when specifying GWC. DEC GKS uses the *current* normalization transformation to transform the fixed point from WC 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.

Description

The ACCUMULATE TRANSFORMATION MATRIX 3 function 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 order of transformation is:

1. Specified input matrix
2. Scale (relative to the specified fixed point)
3. Rotate (relative to the specified fixed point)
4. Shift

See the *DEC GKS User's Guide* for a description of segment transformation and transformation matrixes.

See Also

EVALUATE TRANSFORMATION MATRIX 3
INSERT SEGMENT 3
SET SEGMENT TRANSFORMATION 3
Example 7–1 for a program example using the ACCUMULATE TRANSFORMATION MATRIX function

EVALUATE TRANSFORMATION MATRIX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GEVTM (X0, Y0, DX, DY, PHI, SCX, SCY, SW, MO)

Argument	Data Type	Access	Description
X0, Y0	Real	Read	Fixed point for rotation.
DX, DY	Real	Read	Shift vector. To avoid translation, pass the value 0.0 for these arguments.
PHI	Real	Read	Rotation angle, in radians (360 degrees = 2*pi). To avoid rotation, pass the value 0.0.0 for this argument.
SCX, SCY	Real	Read	Scale factors. To avoid scaling, pass the value 1.0 for these arguments.
SW	Integer (constant)	Read	Flag for WC or NDC values.
MO(2,3)	Array of reals	Write	Transformation matrix that results from evaluating the scaling, rotation, and shifting component values. This value can be used as an argument to the SET SEGMENT TRANSFORMATION or INSERT SEGMENT function to establish a segment transformation. Declare this argument as a 6-element array, as follows: $\begin{bmatrix} MO(1,1) & MO(1,2) & MO(1,3) \\ MO(2,1) & MO(2,2) & MO(2,3) \end{bmatrix}$

Constants

Defined Argument	Constant	Description
SW	GWC	The fixed point and shift vectors are WC values.
	GNDC	The fixed point and shift vectors are NDC values.

Note

Use caution when specifying GWC. DEC GKS uses the *current* normalization transformation to transform the fixed point from WC 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.

Description

The EVALUATE TRANSFORMATION MATRIX function accepts scaling, rotation, and translation component values, and then writes a transformation matrix to the last argument of the function. This function can be used to construct the transformation that can be used as an argument to SET SEGMENT TRANSFORMATION to establish a segment transformation.

The order of transformation is:

1. Scale (relative to the specified fixed point)
2. Rotate (relative to the specified fixed point)
3. Shift

Segment transformation and transformation matrixes are described in the *DEC GKS User's Guide*.

See Also

ACCUMULATE TRANSFORMATION MATRIX
INSERT SEGMENT
SET SEGMENT TRANSFORMATION
Example 7–2 for a program example using the EVALUATE TRANSFORMATION MATRIX function

EVALUATE TRANSFORMATION MATRIX 3

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GEVTM3 (X0, Y0, Z0, DX, DY, DZ, ROTX, ROTY, ROTZ, SCX, SCY, SCZ, SW, MO)

Argument	Data Type	Access	Description
X0, Y0, Z0	Real	Read	Fixed point for rotation.
DX, DY, DZ	Real	Read	Shift vector. To avoid translation, pass the value 0.0.0 for all three arguments.
ROTX, ROTY, ROTZ	Real	Read	Rotation angle, in radians (360 degrees = 2*pi). To avoid rotation, pass the value 0 for all three arguments.
SCX, SCY, SCZ	Real	Read	Scale factors. To avoid scaling, pass the value 1.0 for all three arguments.
SW	Integer (constant)	Read	Flag for WC or NDC values.
MO(3,4)	Array of reals	Write	Transformation matrix that results from evaluating the scaling, rotation, and shifting component values. This value can be used as an argument to the SET SEGMENT TRANSFORMATION 3 or INSERT SEGMENT 3 function to establish a segment transformation. Declare this argument as a 12-element array, as follows: $\begin{bmatrix} MO(1,1) & MO(1,2) & MO(1,3) & MO(1,4) \\ MO(2,1) & MO(2,2) & MO(2,3) & MO(2,4) \\ MO(3,1) & MO(3,2) & MO(3,3) & MO(3,4) \end{bmatrix}$

Constants

Defined Argument	Constant	Description
SW	GWC	The fixed point and shift vectors are WC values.
	GNDC	The fixed point and shift vectors are NDC values.

Note

Use caution when specifying GWC. DEC GKS uses the *current* normalization transformation to transform the fixed point from WC 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.

Description

The EVALUATE TRANSFORMATION MATRIX 3 function accepts scaling, rotation, and translation component values, and then writes a transformation matrix to the last argument of the function. This function can be used to construct the transformation that can be used as an argument to SET SEGMENT TRANSFORMATION 3 to establish a segment transformation.

The order of transformation is:

1. Scale (relative to the specified fixed point)
2. Rotate (relative to the specified fixed point)
3. Shift

Segment transformation and transformation matrixes are described in the *DEC GKS User's Guide*.

See Also

ACCUMULATE TRANSFORMATION MATRIX 3
INSERT SEGMENT 3
SET SEGMENT TRANSFORMATION 3
Example 7–2 for a program example using the EVALUATE TRANSFORMATION MATRIX function

EVALUATE VIEW MAPPING MATRIX 3

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GEVMM3 (VWWNLM, VPORT, PROJ, PRPX, PRPY, PRPZ, VPD, FPD, BPD, ERRIND, VMM)

Argument	Data Type	Access	Description
VWWNLM(4)	Array of reals	Read-only	Window limits in view reference coordinate (VRC) points
VPORT(6)	Array of reals	Read	Projection viewport limits in NPC points, in the order UMIN, UMAX, UMIN, VMAX, VMIN, VMAX
PROJ	Integer (constant)	Read	Projection type
PRPX, PRPY, PRPZ	Real	Read	Projection reference point in VRC points
VPD	Real	Read	View plane distance in VRC points
FPD	Real	Read	Front plane distance in VRC points
BPD	Real	Read	Back plane distance in VRC points
ERRIND	Integer	Write	Error indicator
VMM(4,4)	Array of reals	Write	View mapping matrix:

$$\begin{bmatrix} VMM(1,1) & VMM(1,2) & VMM(1,3) & VMM(1,4) \\ VMM(2,1) & VMM(2,2) & VMM(2,3) & VMM(2,4) \\ VMM(3,1) & VMM(3,2) & VMM(3,3) & VMM(3,4) \\ VMM(4,1) & VMM(4,2) & VMM(4,3) & VMM(4,4) \end{bmatrix}$$

Constants

Defined Argument	Constant	Description
PROJ	GPARL	Parallel projection
	GPERS	Perspective projection

Description

The EVALUATE VIEW MAPPING MATRIX 3 function returns the view mapping matrix for a specified set of input view parameters, which can be passed as input to the SET VIEW REPRESENTATION 3 function. The view mapping matrix in the view representation transforms the GKS coordinate system from VRC points to NPC points.

To create the view mapping matrix, use the following procedure:

- Specify window limits (view window) within VRC space in the order UMIN, UMAX, VMIN, VMAX.

  These restrictions apply:

  UMIN < UMAX
  VMIN < VMAX

  The resulting view window is a rectangular region on the view plane with sides parallel to the U- and V-axes.

- Specify projection viewport limits (view clipping limits) within NPC space in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.

  These restrictions apply:

  XMIN < XMAX
  YMIN < YMAX
  ZMIN $\leq$ ZMAX
  XMIN, XMAX, YMIN, YMAX, ZMIN, and ZMAX must be in the range [0,1], inclusive.

The view clipping limits form a rectangular parallelepiped in NPC space with its edges parallel to the NPC axes. Although the NPC system conceptually extends beyond [0,1] $\times$ [0,1] $\times$[0,1], the view clipping limits are located in the closed unit cube [0,1] $\times$ [0,1] $\times$ [0,1] in NPC space.

The view, back, and front planes are parallel to the UV plane of the VRC system. They are specified as N coordinate values in the three plane arguments to this function.

The front and back plane values specify the front and back of the view volume. Conceptually, the VRC system is oriented, because the VRC points result from the view orientation transformation. (See the EVALUATE VIEW ORIENTATION MATRIX 3 function.) Therefore, the front plane should not be positioned behind the back plane.

The following restrictions apply to the view, front, and back planes:

- Back plane distance < front plane distance
- Back plane distance = front plane distance, if ZMIN = ZMAX
- The N coordinate of the PRP $\neq$ view plane distance
- For projection type = PERSPECTIVE, the N coordinate > front plane distance and < back plane distance

If the view mapping parameters are consistent and well defined (that is, if they conform to the specified rules and restrictions), a call to this function returns the 4 $\times$ 4 view mapping matrix. Otherwise, a nonzero error indicator is returned.

See Also

EVALUATE VIEW ORIENTATION MATRIX 3
SET VIEW REPRESENTATION 3

EVALUATE VIEW ORIENTATION MATRIX 3

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GEVOM3 (VRFPX, VRFPY, VRFPZ, VPNX, VPNY, VPNZ, VUPX, VUPY, VUPZ, SW, ERRIND, VM)

Argument	Data Type	Access	Description
VRFPX, VRFPY, VRFPZ	Real	Read	View reference point
VPNX, VPNY, VPNZ	Real	Read	View plane normal vector
VUPX, VUPY, VUPZ	Real	Read	View up vector
SW	Integer (constant)	Read	Flag indicating WC or NDC values
ERRIND	Integer	Write	Error indicator
VM(4,4)	Array of reals	Write	View orientation matrix:

$$\begin{bmatrix} VM(1,1) & VM(1,2) & VM(1,3) & VM(1,4) \\ VM(2,1) & VM(2,2) & VM(2,3) & VM(2,4) \\ VM(3,1) & VM(3,2) & VM(3,3) & VM(3,4) \\ VM(4,1) & VM(4,2) & VM(4,3) & VM(4,4) \end{bmatrix}$$

Constants

Defined Argument	Constant	Description
SW	GWC	The view reference point, view plane normal vector, and view up vectors are WC values.
	GNDC	The view reference point, view plane normal vector, and view up vectors are NDC values.

Note

Use caution when specifying GWC. DEC GKS uses the *current* normalization transformation to transform the fixed point from WC 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.

Description

The EVALUATE VIEW ORIENTATION MATRIX 3 function provides for three-dimensional translation and rotation of axes. This function returns a view orientation matrix, which can be passed as input to the SET VIEW REPRESENTATION 3 function. The view orientation matrix in the view representation transforms the GKS coordinate system from WC points to VRC points.

The specified **view reference point** is a three-dimensional point that defines the origin of the VRC system.

The specified **view plane normal** is a three-dimensional vector relative to the view reference point. It defines the N-axis of the VRC system, which is the third axis of the system. The **view reference plane** is the plane in WC points that contains the view reference point and is perpendicular to the view plane normal.

The specified **view up vector** is a three-dimensional vector relative to the view reference point. It is projected onto the view reference plane through a projection parallel to the view plane normal. The projection of the view up vector onto the view reference plane determines the V-axis of the VRC system.

These restrictions apply to the specified values:

- View up vector and view plane normal are not parallel; therefore, the view coordinates can be established.
- The length of view up vector is greater than 0.
- The length of view plane normal is greater than 0.

If the view orientation parameters are consistent and well defined (that is, if they conform to the specified rules and restrictions), a call to this function returns the three-dimensional (4×4) view orientation matrix. Otherwise, a nonzero error indicator is returned.

See Also

SET VIEW REPRESENTATION 3

SELECT NORMALIZATION TRANSFORMATION

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSELNT (TNR)

Argument	Data Type	Access	Description
TNR	Integer	Read	Normalization transformation number

Description

The SELECT NORMALIZATION TRANSFORMATION function sets the *normalization transformation number* entry in the GKS state list as the current transformation, and uses the associated window and viewport to transform points from the WC system to the NDC system for subsequent output generation.

To set or reset windows and viewports associated with a transformation number, pass the normalization transformation number to SET WINDOW and SET VIEWPORT. After selecting this number, any subsequent calls to output functions use the window and viewport associated with this number.

By default, DEC GKS uses the unity normalization transformation number 0. Use the default when you want to map the default normalization window to the default NDC viewport.

See Also

SET VIEWPORT
SET VIEWPORT 3
SET WINDOW
SET WINDOW 3
Example 7–3 for a program example using the SELECT NORMALIZATION TRANSFORMATION function

SET CLIPPING INDICATOR

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSCLIP (CLSW)

Argument	Data Type	Access	Description
CLSW	Integer (constant)	Read	Clipping flag

Constants

Defined Argument	Constant	Description
CLSW	GNCLIP	Clipping disabled
	GCLIP	Clipping enabled

Description

The SET CLIPPING INDICATOR function enables or disables clipping of the image at the normalization viewport boundary by setting the clipping flag in the 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.

See Also

INSERT SEGMENT
Example 7–3 for a program example using the SET CLIPPING INDICATOR function

SET VIEW INDEX

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSVWI (VIEWI)

Argument	Data Type	Access	Description
VIEWI	Integer	Read	View index number. The default value is 0.

Description

The SET VIEW INDEX function sets the value of the current view index in the GKS state list. This index is bound to all output primitives. This function associates the view representation of the specified view bundle table entry with all subsequently defined output primitives. The entry contains the following:

- View orientation matrix
- View mapping matrix
- View clipping limits
- XY clipping indicator
- Back clipping indicator
- Front clipping indicator

You can create and change view table indexes and associated table entries with the SET VIEW REPRESENTATION 3 function.

See Also

SET VIEW REPRESENTATION 3

SET VIEW REPRESENTATION 3

Operating States

WSOP, WSAC, SGOP

Syntax

GSVWR3 (WKID, PVWI, VOM, VMM, VCLIP, XYCLIP, BKCLIP, FRCLIP)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
PVWI	Integer	Read	View index
VOM(16)	Array of reals	Read	View orientation matrix
VMM(16)	Array of reals	Read	View mapping matrix
VCLIP(6)	Array of reals	Read	View clipping volume in NPC points, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX
XYCLIP, BKCLIP, FRCLIP	Integer (constant)	Read	XY, front, and back clipping indicators

Constants

Defined Argument	Constant	Description
XYCLIP, BKCLIP, FRCLIP	GNCLIP	Clipping disabled
	GCLIP	Clipping enabled

Description

The SET VIEW REPRESENTATION 3 function modifies the specified view table entries. View changes are applied when the display is updated.

The clipping indicators control whether the planes defined by the clipping limits are active or inactive. If a clipping indicator is turned on, NPC data is clipped at the corresponding plane defined by the clip limits. If a clipping indicator is off, the NPC data is not clipped at the plane. Instead it is allowed to extend through the clip plane until it is conceptually clipped at the NPC system boundary.

SET VIEW TRANSFORMATION INPUT PRIORITY

Operating States

WSOP, WSAC, SGOP

Syntax

GSVTIP (WKID, VIEWI, RFVWIX, RELPRI)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
VIEWI	Integer	Read	View index
RFVWIX	Integer	Read	Reference view index
RELPRI	Integer (constant)	Read	Relative priority

Constants

Defined Argument	Constant	Description
RELPRI	GHIGHR	Next higher priority
	GLOWR	Next lower priority

Description

The SET VIEW TRANSFORMATION INPUT PRIORITY function sets the view transformation input priority of the specified views.

View transformation input priority determines which view transformation is selected to map locator and stroke points from NPC to NDC points. You specify whether the first number is of the next higher or lower priority than the reference number. If you specify lower, the first number is placed directly behind the reference number in the sequential priority list. If you specify higher, DEC GKS places the first number directly in front of the reference number in the sequential priority list. By default, the view representation for view index 0 has the highest view transformation input priority.

If the view index and the reference view index are the same, this function has no effect.

SET VIEWPORT

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSVP (TNR, XMIN, XMAX, YMIN, YMAX)

Argument	Data Type	Access	Description
TNR	Integer	Read	Normalization transformation number
XMIN, XMAX, YMIN, YMAX	Real	Read	Viewport limits in NDC points. Make sure the X and Y values are located within the default normalization viewport boundaries.

Description

The SET VIEWPORT function specifies the viewport limits for the specified normalization transformation.

The normalization transformation maps output primitives and geometric attributes from WC units to NDC units. This mapping is defined by specifying a rectangle in WC points (the normalization window) that is to be mapped to a specified rectangle in NDC points (the normalization viewport). If the two rectangles do not have the same aspect ratios, mapping is not uniform.

SET VIEWPORT modifies the X and Y components of the specified normalization viewport. By default, all normalization transformations have their windows set to [0,1] in X and Y, and their viewports set to [0,1] in X and Y.

See Also

SELECT NORMALIZATION TRANSFORMATION
SET VIEWPORT INPUT PRIORITY
SET WINDOW
Example 7–3 for a program example using the SET VIEWPORT function

SET VIEWPORT 3

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSV3 (TNR, VPLIM)

Argument	Data Type	Access	Description
TNR	Integer	Read	Normalization transformation number
VPLIM(6)	Array of reals	Read	Viewport limits in NDC points, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX. Make sure the X, Y, and Z values are located within the default normalization viewport boundaries.

Description

The SET VIEWPORT 3 function specifies the viewport limits for the specified normalization transformation.

The normalization transformation maps output primitives and geometric attributes from WC units to NDC units. This mapping is defined by specifying a rectangular parallelepiped in WC points (the normalization window) that is to be mapped to a specified rectangular parallelepiped in NDC points (the normalization viewport). If the two parallelepipeds do not have the same aspect ratios, mapping is not uniform.

SET VIEWPORT 3 modifies the X, Y, and Z components of the specified normalization viewport. By default, all normalization transformations have their windows set to [0,1] in X, Y, and Z; and their viewports set to [0,1] in X, Y, and Z.

See Also

SELECT NORMALIZATION TRANSFORMATION
SET VIEWPORT INPUT PRIORITY
SET WINDOW 3
Example 7–3 for a program example using the SET VIEWPORT function

SET VIEWPORT INPUT PRIORITY

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSVPIP (TNR, RTNR, RELPRI)

Argument	Data Type	Access	Description
TNR	Integer	Read	Normalization transformation number
RTNR	Integer	Read	Reference normalization transformation number
RELPRI	Integer (constant)	Read	Relative priority

Constants

Defined Argument	Constant	Description
RELPRI	GHIGHR	Higher priority
	GLOWER	Lower priority

Description

The SET VIEWPORT INPUT PRIORITY function sets the viewport input priority of the specified normalization transformation.

Viewport input priority determines which normalization transformation is selected to map locator and stroke points from NDC points to WC points. By default, the normalization transformations are ordered in a sequential list so that transformation number 0 has the highest viewport input priority and transformation number 255 has the lowest. If you specify HIGHER priority, DEC GKS places the first number directly in front of this reference number in the sequential priority list. If you specify LOWER priority, the first number is placed directly behind this reference number in the sequential priority list.

If the normalization transformation number and the reference normalization transformation numbers are the same, this function has no effect.

See Also

GET LOCATOR
GET STROKE
REQUEST LOCATOR
REQUEST STROKE
SAMPLE LOCATOR
SAMPLE STROKE
SELECT NORMALIZATION TRANSFORMATION
SET WINDOW

SET WINDOW

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSWN (TNR, XMIN, XMAX, YMIN, YMAX)

Argument	Data Type	Access	Description
TNR	Integer	Read	Normalization transformation number
XMIN, XMAX, YMIN, YMAX	Real	Read	Window limits in WC points

Description

The SET WINDOW function specifies the window limits for the specified normalization transformation.

The normalization transformation maps output primitives and geometric attributes from WC units to NDC units. This mapping is defined by specifying a rectangle in WC points (the normalization window) that is to be mapped to a specified rectangle in NDC points (the normalization viewport). If the two rectangles do not have the same aspect ratios, mapping is not uniform.

SET WINDOW modifies the X and Y components of the specified normalization window. By default, all normalization transformations have their windows set to [0,1] in X and Y; and their viewports set to [0,1] in X and Y.

See Also

SELECT NORMALIZATION TRANSFORMATION
SET VIEWPORT
SET VIEWPORT INPUT PRIORITY
Example 7–3 for a program example using the SET WINDOW function

SET WINDOW 3

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GSW3 (TNR, WNLIM)

Argument	Data Type	Access	Description
TNR	Integer	Read	Normalization transformation number
WNLIM(6)	Array of reals	Read	Window limits in WC points, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, and ZMAX

Description

The SET WINDOW 3 function specifies the window limits for the specified normalization transformation.

The normalization transformation maps output primitives and geometric attributes from WC units to NDC units. This mapping is defined by specifying a rectangular parallelepiped in WC points (the normalization window) that is to be mapped to a specified rectangular parallelepiped in NDC points (the normalization viewport). If the two parallelepipeds do not have the same aspect ratios, mapping is not uniform.

SET WINDOW 3 modifies the X, Y, and Z components of the specified normalization window. By default, all normalization transformations have their windows set to [0,1] in X, Y, and Z; and their viewports set to [0,1] in X, Y, and Z.

See Also

SELECT NORMALIZATION TRANSFORMATION
SET VIEWPORT 3
SET VIEWPORT INPUT PRIORITY
Example 7–3 for a program example using the SET WINDOW function

SET WORKSTATION VIEWPORT

Operating States

WSOP, WSAC, SGOP

Syntax

GSWKVP (WKID, XMIN, XMAX, YMIN, YMAX)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
XMIN, XMAX, YMIN, YMAX	Real	Read	Workstation viewport limits in device coordinate points.

Description

The SET WORKSTATION VIEWPORT function establishes the portion of the workstation surface on which DEC GKS maps the workstation window. Make sure the X and Y values are located within the display surface limits of the specified workstation. Use the function INQUIRE DISPLAY SPACE SIZE to determine the maximum X and Y values of the workstation display surface.

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.

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.

See Also

INQUIRE DISPLAY SPACE SIZE
SET WORKSTATION WINDOW
Example 7–4 for a program example using the SET WORKSTATION VIEWPORT function

SET WORKSTATION VIEWPORT 3

Operating States

WSOP, WSAC, SGOP

Syntax

GSWKV3 (WKID, WKVP)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
WKVP(6)	Array of reals	Read	Workstation viewport limits in device coordinate points, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.

Description

The SET WORKSTATION VIEWPORT 3 function establishes the portion of the workstation surface on which DEC GKS maps the workstation window. Make sure the X, Y, and Z values are located within the display surface limits of the specified workstation. Use the function INQUIRE DISPLAY SPACE SIZE 3 to determine the maximum X, Y, and Z values of the workstation display surface.

The default workstation viewport is the largest cube on the workstation surface, with the origin at the lower left corner furthest from the observer of the display space. 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.

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.

See Also

INQUIRE DISPLAY SPACE SIZE 3
SET WORKSTATION WINDOW 3
Example 7–4 for a program example using the SET WORKSTATION VIEWPORT function

SET WORKSTATION WINDOW

Operating States

WSOP, WSAC, SGOP

Syntax

GSWKWN (WKID, XMIN, XMAX, YMIN, YMAX)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
XMIN, XMAX, YMIN, YMAX	Real	Read	Workstation window limits in NDC points.

Description

The SET WORKSTATION WINDOW function 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 ($[0,1] \times [0,1]$) 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.

See Also

SET WORKSTATION VIEWPORT

SET WORKSTATION WINDOW 3

Operating States

WSOP, WSAC, SGOP

Syntax

GSWKW3 (WKID, WKWN)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
WKWN(6)	Array of reals	Read	Workstation window limits in NDC points, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.

Description

The SET WORKSTATION WINDOW 3 function establishes the portion of the composed picture, in NDC space, 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] $\times$ [0,1] $\times$ [0,1]) onto the largest cube 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.

See Also

SET WORKSTATION VIEWPORT 3

7.6 Program Examples

Example 7–1 illustrates the use of the ACCUMULATE TRANSFORMATION MATRIX function.

Example 7–1 Showing the Cumulative Effect of ACCUMULATE TRANSFORMATION MATRIX

```
C  This program shows how using the ACCUMULATE TRANSFORMATION MATRIX
C  function lets you add transformation components to a previously
C  set transformation.
C
C  NOTE: To keep the example concise, no error checking is performed.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER   DEV_NUM
      INTEGER  HOUSE
      INTEGER IN_CLASS
      INTEGER  LOWER_LEFT_CORNER
      REAL NO_CHANGE
      REAL NULL
      INTEGER NUM_POINTS
      REAL PX (9)
      REAL  PY (9)
      REAL TIME_OUT
      REAL      VECTOR_Y
      INTEGER   WS_ID
      REAL  XFORM_MATRIX (6)

C  Open and activate GKS and the workstation environment.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Set the viewport limits for the specified normalization
C  transformation.

      LOWER_LEFT_CORNER = 1

      CALL GSVP (LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5)

C  This call selects a normalization transformation with the
C  new viewport.

      CALL GSELNT (LOWER_LEFT_CORNER)
      CALL GSCLIP (GNCLIP)

C  Create the segment.
```

(continued on next page)

Example 7–1 (Cont.) Showing the Cumulative Effect of ACCUMULATE TRANSFORMATION MATRIX

```
      HOUSE = 1
      NUM_POINTS = 9
      PX(1) = .4
      PY(1) = .1
      PX(2) = .1
      PY(2) = .1
      PX(3) = .1
      PY(3) = .7
      PX(4) = .4
      PY(4) = .7
      PX(5) = .25
      PY(5) = .9
      PX(6) = .1
      PY(6) = .7
      PX(7) = .4
      PY(7) = .1
      PX(8) = .4
      PY(8) = .7
      PX(9) = .1
      PY(9) = .1

      CALL GCRSG (HOUSE)
      CALL GPL (NUM_POINTS, PX, PY)
      CALL GCLSG ()

C  Release the deferred output.  Wait 3 seconds.

      TIME_OUT = 3.00

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  Shift the house upwards by 1 world coordinate.

      NO_CHANGE = 1.0
      NULL = 0
      VECTOR_Y = .1

      CALL GEVTM (NULL, NULL, NULL, VECTOR_Y, NULL,
     *NO_CHANGE, NO_CHANGE, GWC, XFORM_MATRIX)

C  Transform the segment and update the screen.  Calling SET SEGMENT
C  TRANSFORMATION changes the segment transformation in the segment list,
C  and sets flags in the workstation state list, telling GKS that the
C  display surface is out of date and that an update is necessary.

      CALL GSSGT (HOUSE, XFORM_MATRIX)

C  Calling UPDATE WORKSTATION updates the position of the image on the
C  workstation surface.

      CALL GUWK (WS_ID, GPERFO)

C  Release the deferred output.  Wait 3 seconds.

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)
```

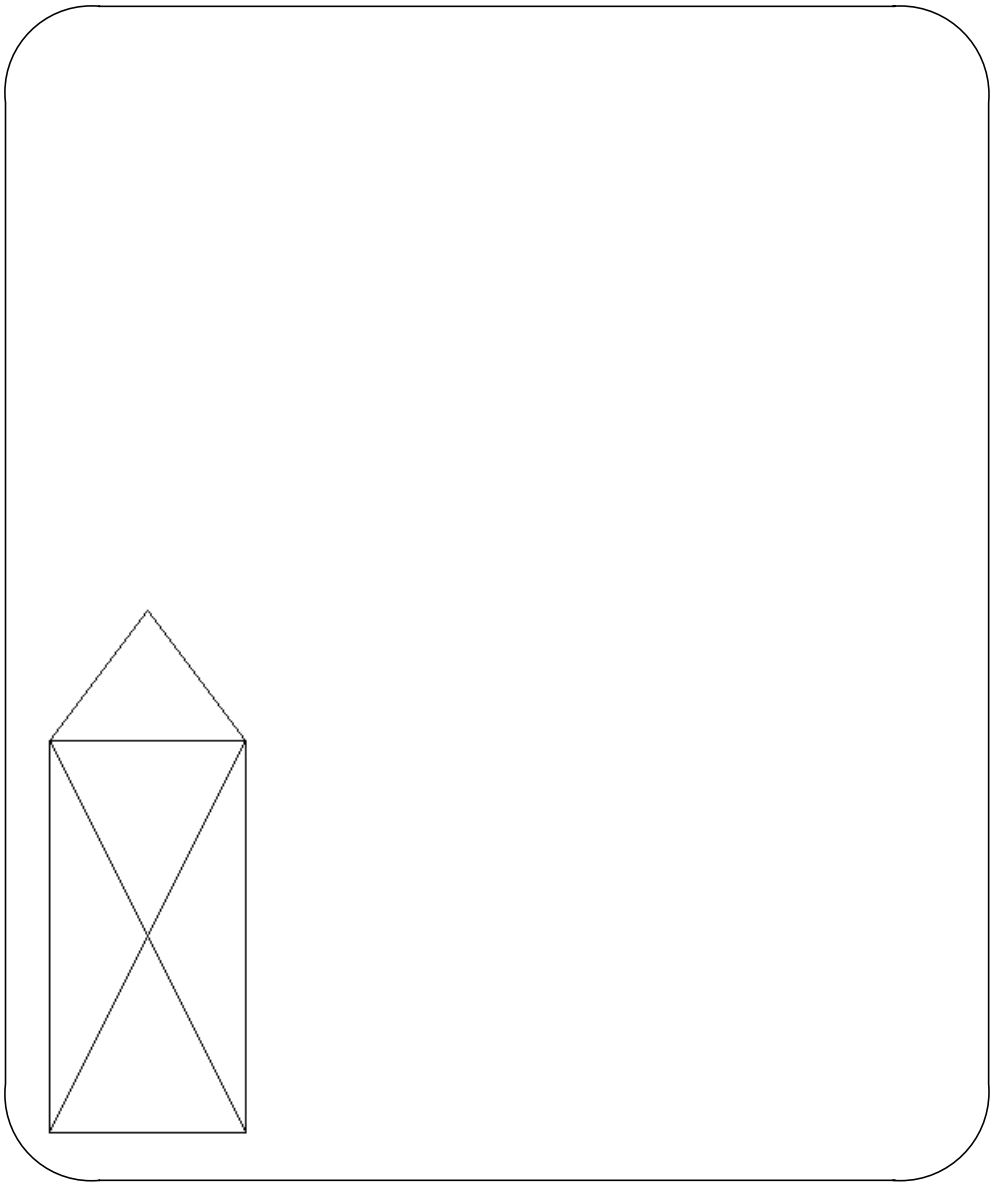
(continued on next page)

Example 7–1 (Cont.) Showing the Cumulative Effect of ACCUMULATE TRANSFORMATION MATRIX

```
C  Using ACCUMULATE TRANSFORMATION MATRIX, you can add transformation
C  components to a previously set transformation.  The house gradually
C  moves upward, one Y world coordinate point at a time.

      CALL GACTM (XFORM_MATRIX, NULL, NULL, NULL, VECTOR_Y,
     *NULL, NO_CHANGE, NO_CHANGE, GWC, XFORM_MATRIX)

C  Transform the segment and update the screen.

      CALL GSSGT (HOUSE, XFORM_MATRIX)
      CALL GUWK (WS_ID, GPERFO)

C  Release the deferred output.  Wait 3 seconds.

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  Again, shift the house upwards by 1 more world coordinate.

      CALL GACTM (XFORM_MATRIX, NULL, NULL, NULL, VECTOR_Y,
     *NULL, NO_CHANGE, NO_CHANGE, GWC, XFORM_MATRIX)

C  Transform the segment.

      CALL GSSGT (HOUSE, XFORM_MATRIX)

C  Update the surface to initiate the change.

      CALL GUWK (WS_ID, GPERFO)

C  Deactivate and close the workstation environment and GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

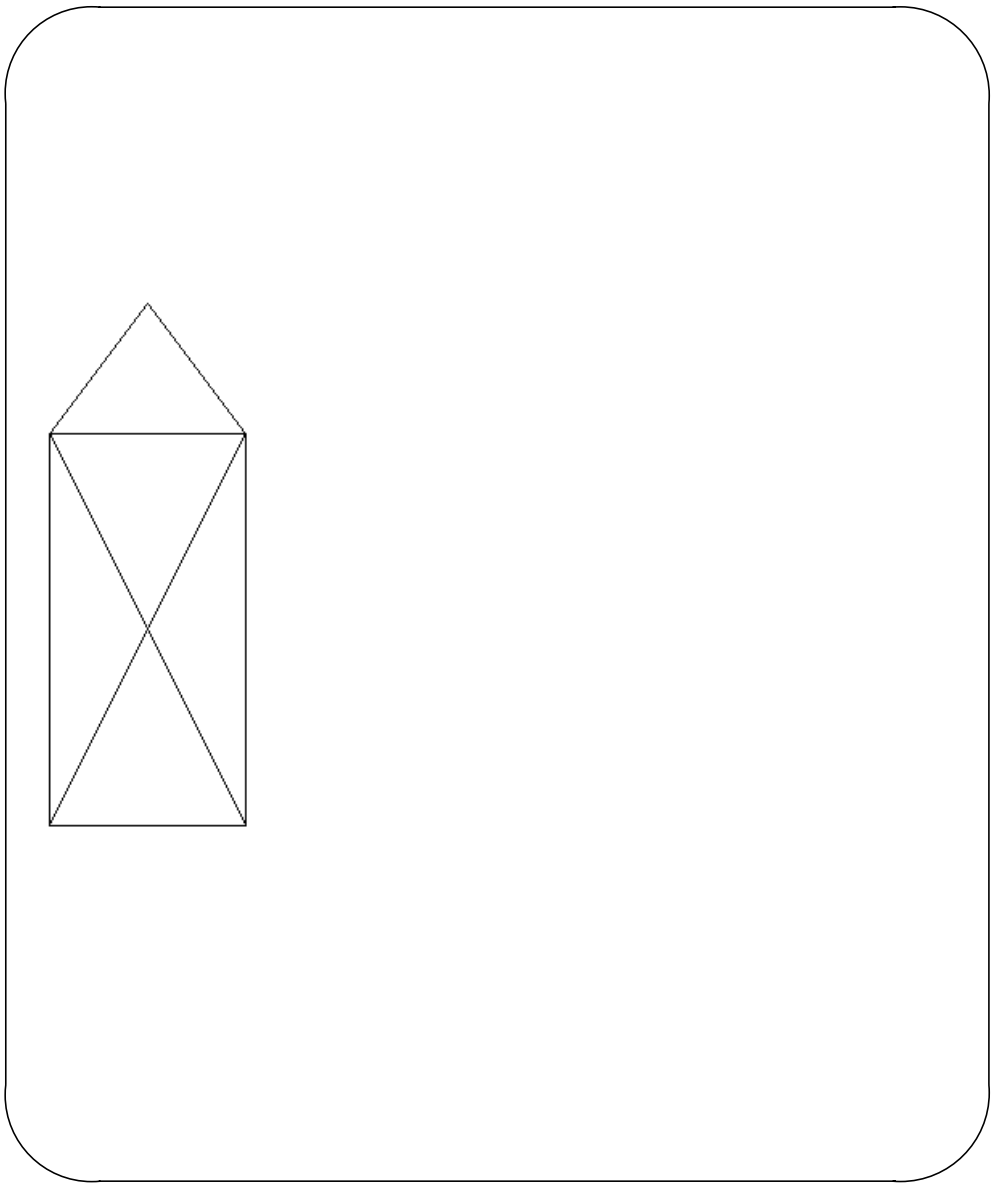
Figure 7–4 and Figure 7–5 show the first and last positions of the house. Each of the four house positions illustrates an added transformation component in the ACCUMULATE TRANSFORMATION MATRIX function.

Figure 7–4 First Transformation Component of ACCUMULATE TRANSFORMATION MATRIX



ZK–4019A–GE

Figure 7–5 Fourth Transformation Component of ACCUMULATE TRANSFORMATION MATRIX



ZK–4022A–GE

Example 7–2 illustrates the use of the EVALUATE TRANSFORMATION MATRIX function.

Example 7–2 The Effects of a Segment Transformation

```
C     This program transforms the house contained in a segment.  The
C     program shows the effects of segment transformation through the
C     use of the EVALUATE TRANSFORMATION MATRIX function.
C
C     NOTE:  To keep the example concise, no error checking is performed.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER DEV_NUM
      REAL FIXED_X
      REAL FIXED_Y
      INTEGER  HOUSE
      INTEGER IN_CLASS
      INTEGER  LOWER_LEFT_CORNER
      INTEGER NUM_POINTS
      REAL PX (9)
      REAL  PY (9)
      REAL      ROTATION
      REAL SCALE_X
      REAL  SCALE_Y
      REAL TIME_OUT
      REAL  VECTOR_X
      REAL VECTOR_Y
      INTEGER   WS_ID
      REAL  XFORM_MATRIX (6)

C  Open and activate GKS and the workstation environments.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Set the viewport limits for the specified normalization
C  transformation.

      LOWER_LEFT_CORNER = 1

      CALL GSVP (LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5)
      CALL GSELNT (LOWER_LEFT_CORNER)
      CALL GSCLIP (GNCLIP)
```

(continued on next page)

Example 7–2 (Cont.) The Effects of a Segment Transformation

```
C  Create the segment.

      HOUSE = 1
      NUM_POINTS = 9
      PX(1) = .4
      PY(1) = .1
      PX(2) = .1
      PY(2) = .1
      PX(3) = .1
      PY(3) = .7
      PX(4) = .4
      PY(4) = .7
      PX(5) = .25
      PY(5) = .9
      PX(6) = .1
      PY(6) = .7
      PX(7) = .4
      PY(7) = .1
      PX(8) = .4
      PY(8) = .7
      PX(9) = .1
      PY(9) = .1

      CALL GCRSG (HOUSE)
      CALL GPL (NUM_POINTS, PX, PY)
      CALL GCLSG ()

C  Release the deferred output.  Wait 5 seconds.

      TIME_OUT = 5.00

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  Rotation equals pi divided by 6 (30 degrees).

      ROTATION = 3.14/6.0

C  You can change the segment transformation that affects the
C  rotation, scaling, and translation components of segment
C  appearance.  The EVALUATE TRANSFORMATION MATRIX call assists in the
C  creation of a new transformation matrix, that will permit you to
C  specify rotation, scaling, and translation values.

      FIXED_X = .25
      FIXED_Y = .9
      SCALE_X = .5
      SCALE_Y = .5
      VECTOR_X = .0
      VECTOR_Y = .2

      CALL GEVTM (FIXED_X, FIXED_Y, VECTOR_X, VECTOR_Y,
     *ROTATION, SCALE_X, SCALE_Y, GWC, XFORM_MATRIX)

C  Transform the segment.

      CALL GSSGT (HOUSE, XFORM_MATRIX)
```

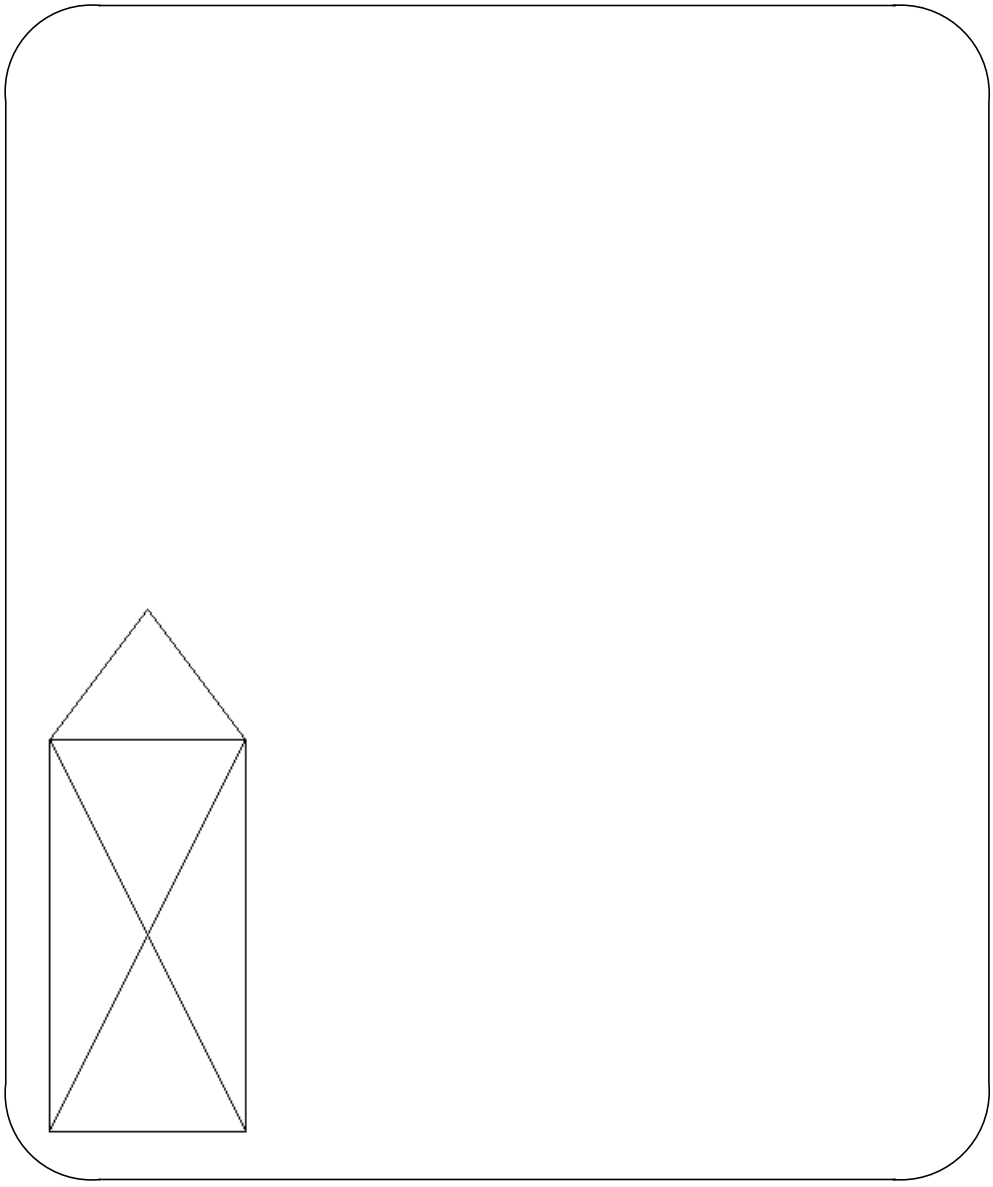
(continued on next page)

Example 7–2 (Cont.) The Effects of a Segment Transformation

```
C  The UPDATE WORKSTATION function updates the position of the image on
C  the workstation surface.  All output not contained in segments is lost.

      CALL GUWK (WS_ID, GPERFO)

C  Deactivate and close the workstation environment and GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 7–6 shows the house before the segment transformation.

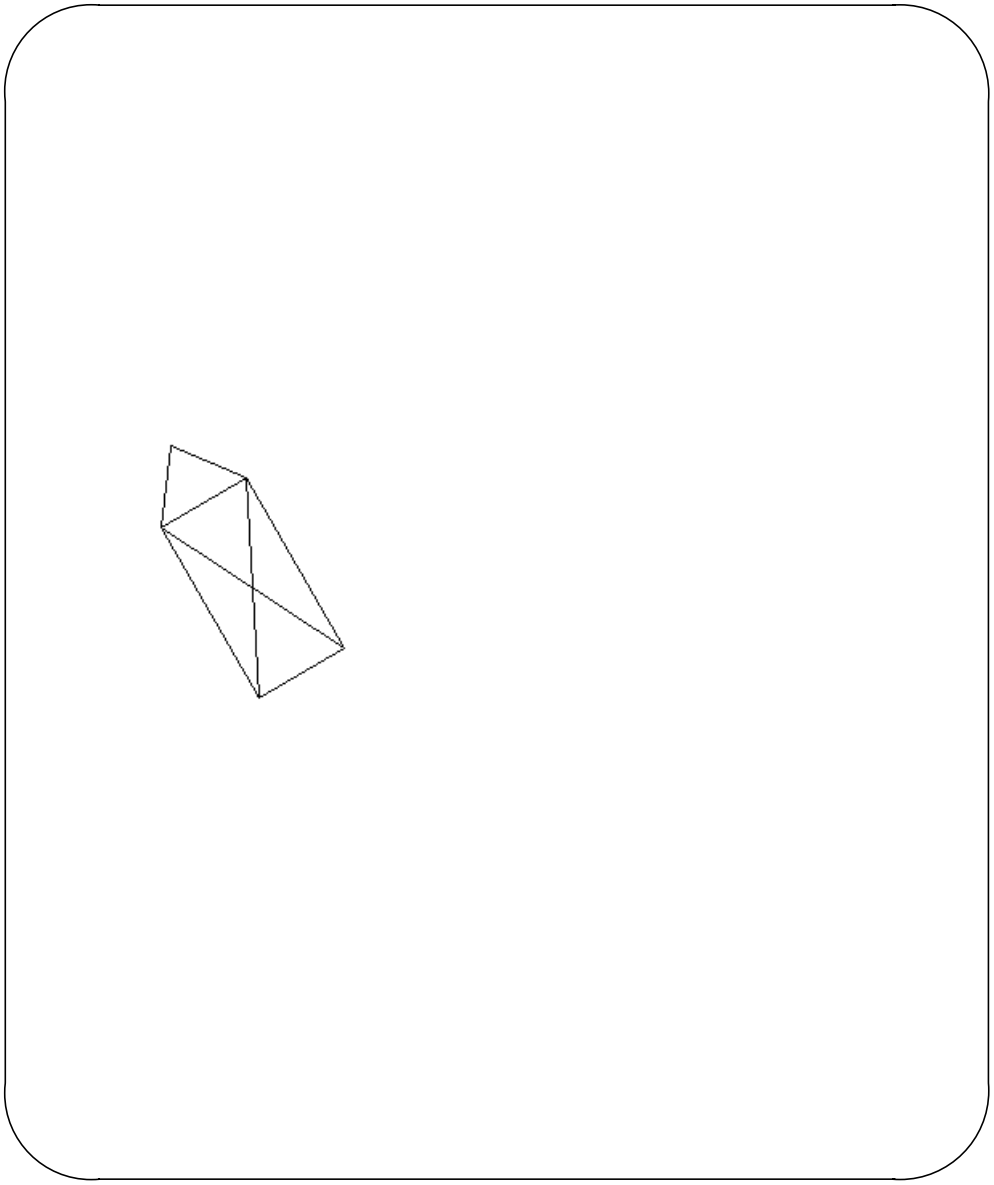
Figure 7–6 Output Prior to Segment Transformation



ZK–4019A–GE

Figure 7–7 shows the house after the effects of the segment transformation.

Figure 7–7 Effect of Segment Transformation



Example 7–3 illustrates the use of the SET CLIPPING INDICATOR function.

Example 7–3 Controlling Clipping at the World Viewport

```
C  This program illustrates the SET CLIPPING INDICATOR function.  This
C  program generates a "tall, thin house" that overlaps the
C  normalization window and viewport.  You can see the overlapping
C  portion if clipping is disabled.
C
C  NOTE: To keep the example concise, no error checking is performed.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER   DEV_NUM
      INTEGER HALF
      INTEGER   IN_CLASS
      INTEGER LOW_LEFT_CORNER
      INTEGER NUM_POINTS
      REAL PX (9)
      REAL PX_2 (5)
      REAL PY (9)
      REAL      PY_2 (5)
      REAL      TIME_OUT
      INTEGER   WS_ID

C  Open and activate GKS and the workstation environment.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Outlining the default world window results in the outlining of
C  the NDC plane, the workstation window, and the workstation
C  viewport, by default.

      PX_2 (1) = 0.0
      PY_2 (1) = 0.0
      PX_2 (2) = 0.5
      PY_2 (2) = 0.0
      PX_2 (3) = 0.5
      PY_2 (3) = 0.5
      PX_2 (4) = 0.0
      PY_2 (4) = 0.5
      PX_2 (5) = 0.0
      PY_2 (5) = 0.0

      CALL GPL (5, PX_2, PY_2)

C  Set the normalization window to be half of the default
C  plane and assign the normalization transformation low_left_corner.
C  The house is cut in half by the window boundary.  Set the viewport
C  to be the lower left corner of the NDC space and assign this
C  normalization transformation the value 1.

      HALF = 1
      LOW_LEFT_CORNER = 1

      CALL GSWN (HALF, 0.0, 0.9, 0.0, 0.5)
      CALL GSVP (LOW_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5)
```

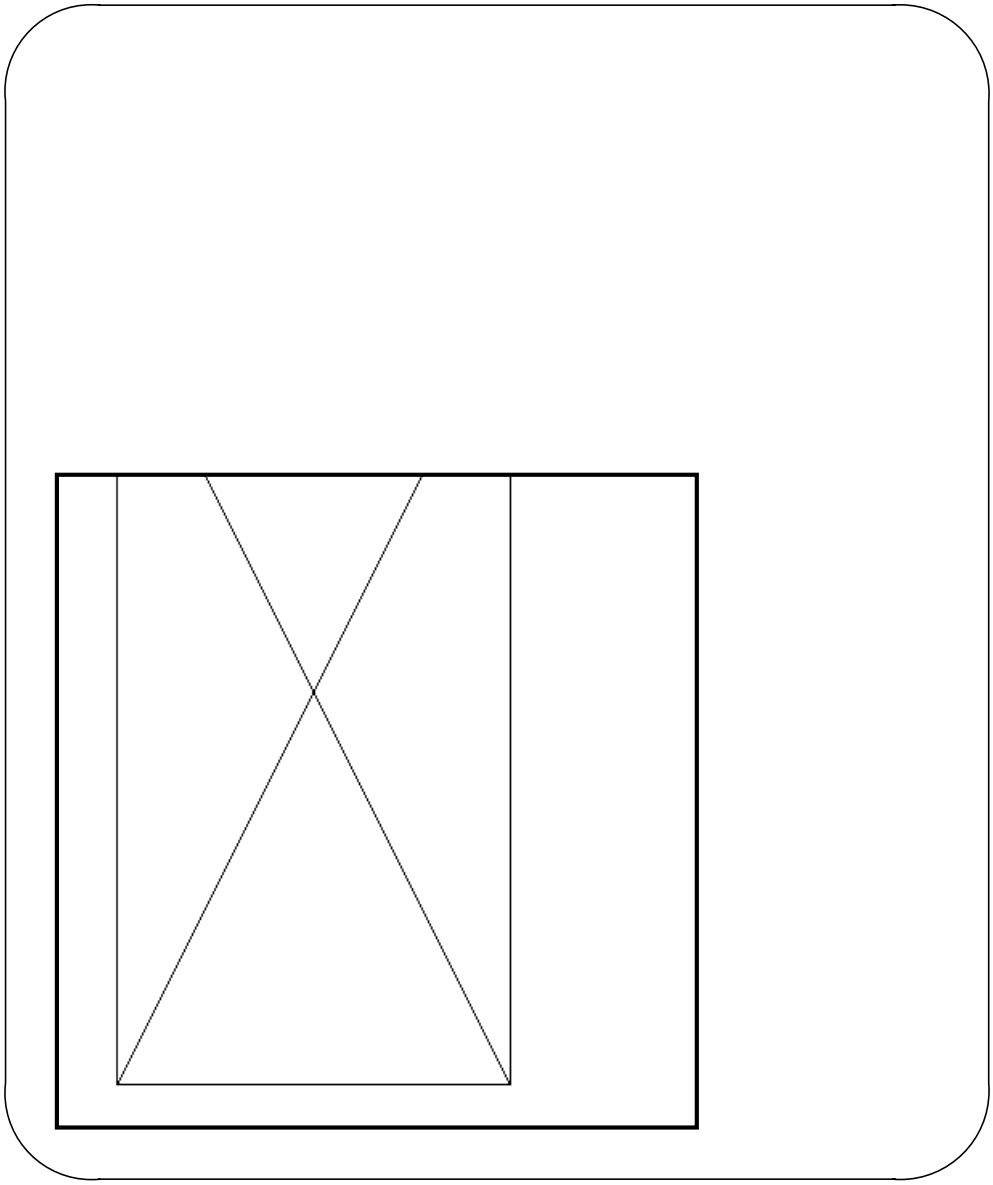
(continued on next page)

Example 7–3 (Cont.) Controlling Clipping at the World Viewport

```
C  Select the normalization transformation number 1 which has a
C  smaller window by half and a viewport that is the lower left
C  corner of the NDC space.  By default, clipping is enabled; you
C  only see half the house.

      NUM_POINTS = 9
      PX (1) = .4
      PY (1) = .1
      PX (2) = .1
      PY (2) = .1
      PX (3) = .1
      PY (3) = .7
      PX (4) = .4
      PY (4) = .7
      PX (5) = .25
      PY (5) = .9
      PX (6) = .1
      PY (6) = .7
      PX (7) = .4
      PY (7) = .1
      PX (8) = .4
      PY (8) = .7
      PX (9) = .1
      PY (9) = .1

      CALL GSELNT (LOW_LEFT_CORNER)
      CALL GPL (NUM_POINTS, PX, PY)

C  Release the deferred output. Wait 5 seconds.

      TIME_OUT = 5.00

      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  Once you disable clipping and redraw the picture, GKS maps the
C  entire house to NDC space and eventually the workstation surface.

      CALL GSCLIP (GNCLIP)

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 GPL (NUM_POINTS, PX, PY)

C  Release the deferred output. Wait 5 seconds.

      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  Deactivate and close the workstation environment and GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 7–8 illustrates the house while clipping is enabled.

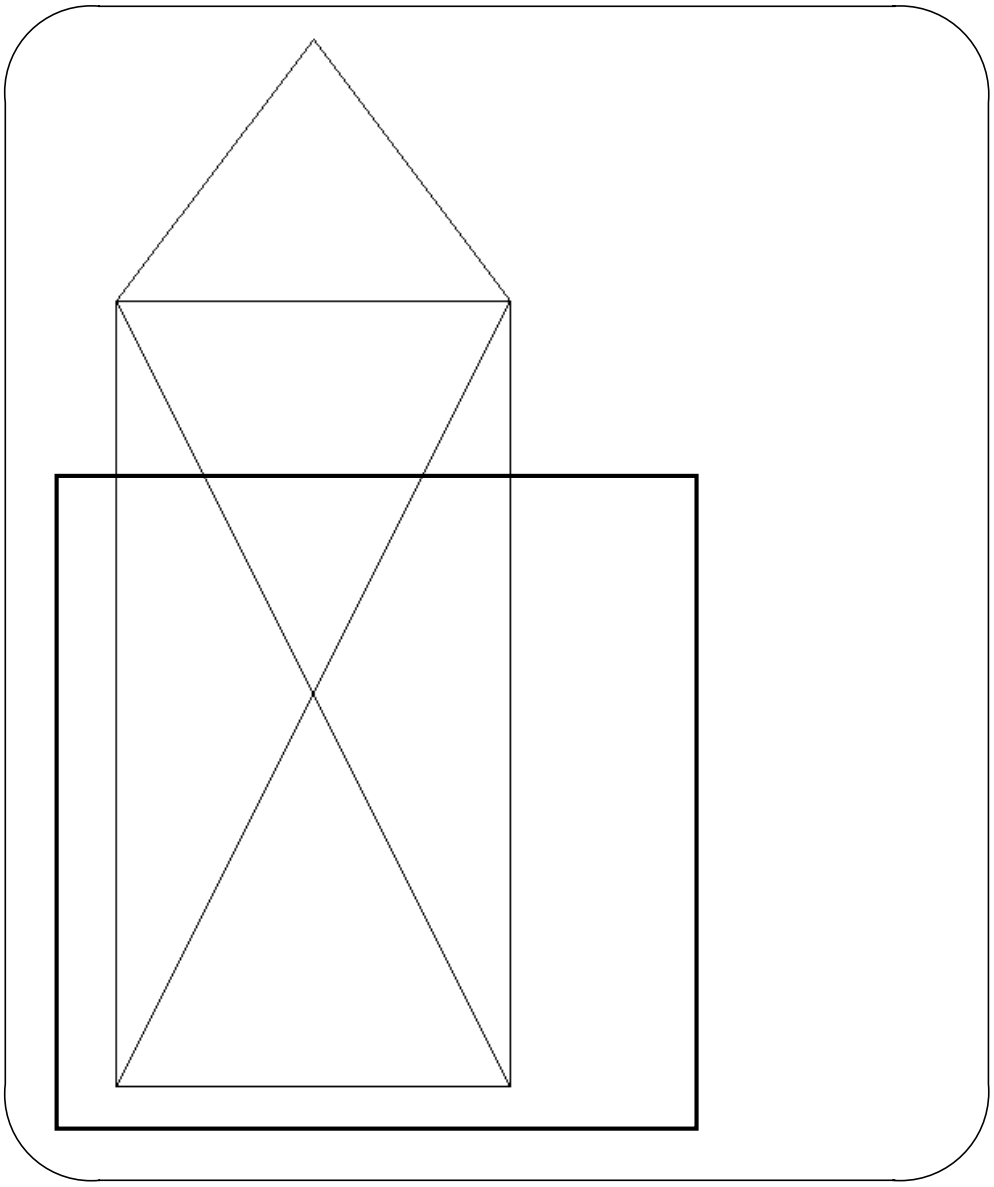
Figure 7–8 SET CLIPPING INDICATOR with Clipping Enabled



ZK–4027A–GE

Figure 7–9 illustrates the house while clipping is disabled. The house overlaps the normalization window and viewport.

Figure 7–9 SET CLIPPING INDICATOR with Clipping Disabled



ZK–4028A–GE

Example 7–4 illustrates the use of the SET WORKSTATION VIEWPORT function.

Example 7–4 Establishing a Workstation Viewport

```
C  This program illustrates the SET WORKSTATION VIEWPORT function.
C  This program uses the default normalization transformations,
C  generates a "tall, thin house," updates the screen,
C  changes the workstation viewport to the lower left
C  corner of the display surface, and generates the output again.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      REAL  DEVICE_MAX_X
      REAL DEVICE_MAX_Y
      INTEGER   DEV_NUM
      INTEGER ERROR
      INTEGER  HOUSE
      INTEGER   IN_CLASS
      INTEGER METERS
      INTEGER   NOT_USED
      INTEGER NUM_POINTS
      REAL PX (9)
      REAL PX_2 (5)
      REAL PY (9)
      REAL PY_2 (5)
      INTEGER RASTER_X
      INTEGER RASTER_Y
      REAL      TIME_OUT
      INTEGER   WKTR
      INTEGER WS_ID

C  Open and activate GKS and the workstation environments.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Outlining the default world window results in outlining
C  the NDC plane, the workstation window, and the workstation
C  viewport, by default.

      PX_2 (1) = 0.0
      PY_2 (1) = 0.0
      PX_2 (2) = 1.0
      PY_2 (2) = 0.0
      PX_2 (3) = 1.0
      PY_2 (3) = 1.0
      PX_2 (4) = 0.0
      PY_2 (4) = 1.0
      PX_2 (5) = 0.0
      PY_2 (5) = 0.0

      CALL GPL (5, PX_2, PY_2)

C  This code assumes the default normalization transformation.
C  GKS maps the default window to the default viewport, then
C  maps the default workstation window to the default viewport.

      HOUSE = 1
      NUM_POINTS = 9

      CALL GCRSG (HOUSE)
```

(continued on next page)

Example 7–4 (Cont.) Establishing a Workstation Viewport

```
      PX (1) = 0.4
      PY (1) = 0.1
      PX (2) = 0.1
      PY (2) = 0.1
      PX (3) = 0.1
      PY (3) = 0.7
      PX (4) = 0.4
      PY (4) = 0.7
      PX (5) = 0.25
      PY (5) = 0.9
      PX (6) = 0.1
      PY (6) = 0.7
      PX (7) = 0.4
      PY (7) = 0.1
      PX (8) = 0.4
      PY (8) = 0.7
      PX (9) = 0.1
      PY (9) = 0.1

      CALL GPL (NUM_POINTS, PX, PY)
      CALL GCLSG ()

C  Release the deferred output.  Wait 5 seconds.

      TIME_OUT = 5.00

      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  The function INQUIRE DISPLAY SPACE SIZE returns the maximum X
C  and Y display surface coordinates in the arguments device_max_x
C  and device_max_y.

      CALL GQDSP (GDECW, ERROR, METERS, DEVICE_MAX_X,
     *DEVICE_MAX_Y, RASTER_X, RASTER_Y)

C  The function SET WORKSTATION VIEWPORT changes the workstation
C  viewport to the lower left corner of the screen.  The picture
C  being displayed is still the same (the default workstation
C  window), but the space on the workstation surface used to
C  display the same picture has changed.

      CALL GSWKVP (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).

      TIME_OUT = 5.00

      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  Check whether the workstation viewport change required an implicit
C  regeneration (IRG), thereby deleting all information not retained in a
C  segment.

      CALL GQDWKA (GWSDEF, ERROR, NOT_USED, NOT_USED, NOT_USED,
     *NOT_USED, NOT_USED, NOT_USED, WKTR)

      IF (WKTR .EQ. GIRG) THEN

C  Outline the default window again.

        CALL GPL (5, PX_2, PY_2)
      END IF
```

(continued on next page)

Example 7–4 (Cont.) Establishing a Workstation Viewport

```
C  The display pauses for 5 seconds so you can view the house in
C  the new vieport.

      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  Deactivate and close the workstation environment and GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 7–10 illustrates how the house is displayed using the default normalization transformation.

Figure 7–10 Output Using the Default Normalization Transformation

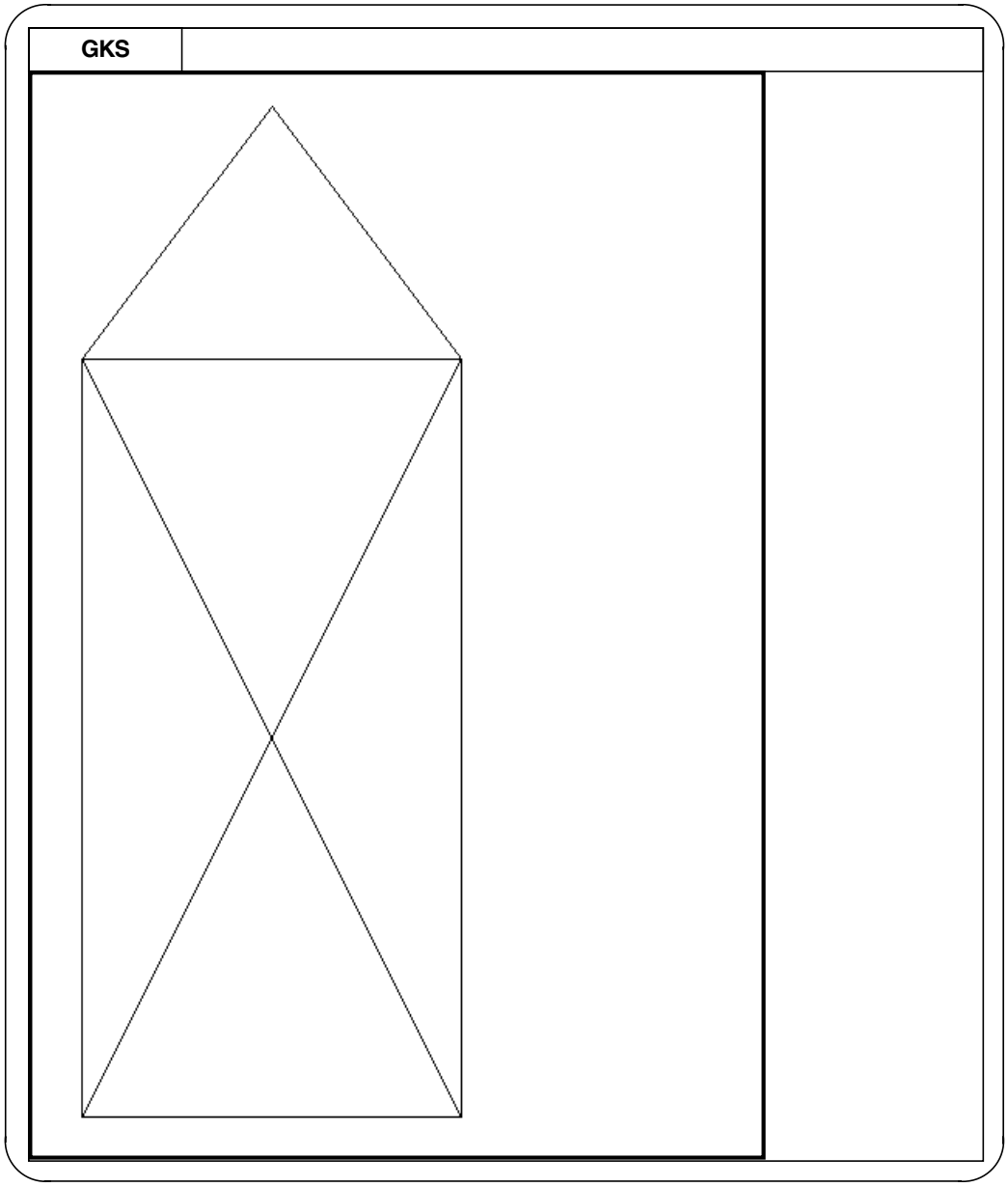
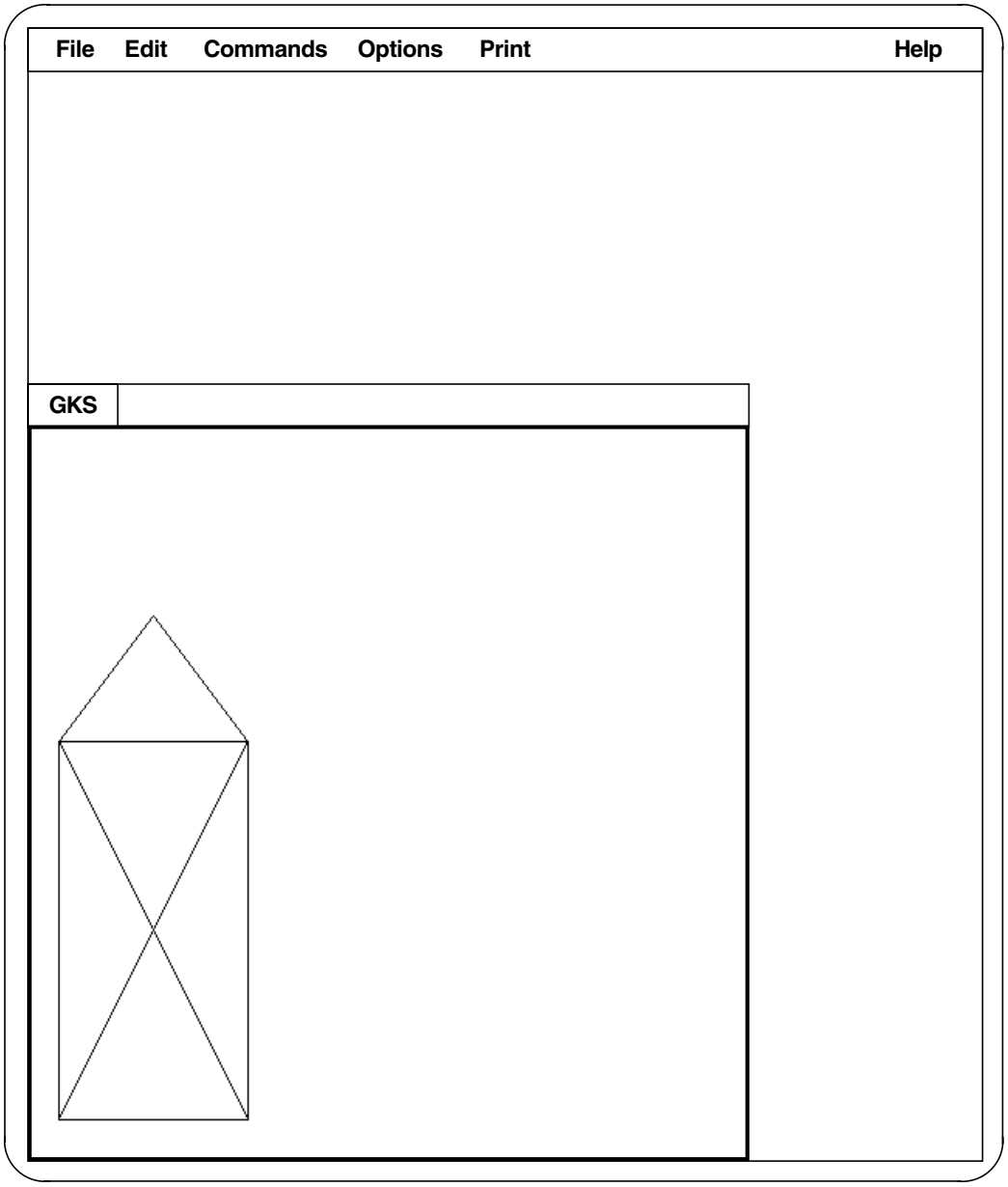


Figure 7–11 illustrates how the house is displayed with changes to the workstation viewport.

Figure 7–11 Output After Changes to the Workstation Viewport



Segment Functions

Insert tabbed divider here. Then discard this sheet.

8
Segment Functions

The DEC GKS segment functions create, manipulate, and delete stored groups of output primitives called **segments**.

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 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.

8.1 Creating, Using, and Deleting Segments

To use segments, your workstation should be one of the categories OUTPUT, OUTIN, MO, or WISS (described in Section 8.2). 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 WSAC (at least one workstation active). When DEC GKS is in this state, you can call CREATE SEGMENT, which creates a segment on all active workstations. The only argument passed to CREATE SEGMENT is the segment name. You use a segment name to identify a particular segment.

After you call CREATE SEGMENT, the DEC GKS operating state changes to SGOP (segment open). Subsequent calls to the DEC GKS output and attribute functions produce primitives stored in the segment on all active workstations. When you have created the desired image, call CLOSE SEGMENT. This call closes the segment, causing the DEC GKS operating state to change back to WSAC.

When you call CREATE SEGMENT, the DEC GKS operating state changes from WSAC to SGOP. SGOP signifies that a segment is open, or being created. Also, calling CREATE SEGMENT establishes the segment state list associated with the segment name, and DEC GKS records the segment name (in the GKS state list) as the name of the currently open segment.

Segments cannot contain other segments; in other words, segments cannot be nested. Therefore, if you call CREATE SEGMENT, you must call CLOSE SEGMENT before you attempt to call CREATE SEGMENT again. Until you call CLOSE SEGMENT, DEC GKS associates all generated output primitives with the name of the open segment. When you call CLOSE SEGMENT, the DEC GKS operating state changes from SGOP back to 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 RENAME SEGMENT. 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 DELETE SEGMENT FROM WORKSTATION, DEC GKS deletes the segment from the specified workstation. If you use DELETE SEGMENT, DEC GKS deletes the specified segment from all workstations storing the segment. If you call CLEAR WORKSTATION, 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, see Chapter 4.

Note

If you store primitives in a segment, and want to be able to change the primitive's appearance elsewhere in the program, you must set the primitive's ASF to be GBUNDL 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, 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 on aspect source flags, see Chapter 6.

8.1.1 Pick Identification

One of the DEC GKS logical input classes is the **pick** input class. Using the function REQUEST PICK, the user can choose a segment, and possibly a portion of the segment, as displayed on the surface of the workstation.

REQUEST PICK returns the segment name and the **pick identifier** of the segment or segment portion chosen by the user. The pick identifier is a numeric output attribute. Like other output attribute values (line type, line width, color, text alignment, and so on), 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.

To see how to use pick identifiers, see Example 9–2 for a program example that calls SET PICK IDENTIFIER.

8.2 Workstations and Segment Storage

When DEC GKS stores a segment on an active OUTPUT, OUTIN, or MO workstation, the method of storage is called workstation dependent segment storage (WDSS). On these workstations, you can control the segment attributes (see Section 8.4), move or alter the shape of the segment using the segment transformation functions (see Section 8.4.4.1), or 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 non-WISS workstation and recall it again. 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 applies an INSERT SEGMENT transformation and then copies all 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 INSERT SEGMENT 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 GWSWIS (value 5) as the workstation type. When you open WISS, you can specify GWSDEF as the connection identifier argument. (If you specify GWSWIS, DEC GKS ignores the connection identifier argument.)

Once you activate the WISS workstation and create segments, you can use the DEC GKS functions ASSOCIATE SEGMENT WITH WORKSTATION, COPY SEGMENT TO WORKSTATION, and INSERT SEGMENT. Example 8–1 shows the difference between ASSOCIATE SEGMENT WITH WORKSTATION and COPY SEGMENT TO WORKSTATION. See Example 8–2 for a program example using INSERT SEGMENT.

8.3 Segments and Surface Update

When you request changes to segment attributes (described in Section 8.4), 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 primitives not stored in segments are lost. You can use the function INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES 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 INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES to determine if the *new frame necessary at update* entry is YES. If you are making many different changes, calling this function each time is inefficient.

You can call INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES 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, call UPDATE WORKSTATION and pass GPERFO as an argument.

Note

If you want to redraw all the segments on the workstation surface regardless of the current status of the new frame flag, you can call REDRAW ALL SEGMENTS ON WORKSTATION. A call to this function is equivalent to a call to UPDATE WORKSTATION while the new frame flag is set to YES, and while passing the argument GPERFO.

Requests for changes to segments may require an implicit regeneration of the screen depending on the capabilities of your device (see Section 8.4 for a complete descriptions of the segment attributes). Table 8–1 describes surface regeneration resulting from changes to segments.

Table 8–1 Surface Regeneration from Changes to Segments

Change	Possible Effect
Segment priority	Calls to the following functions may create 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. • ASSOCIATE SEGMENT WITH WORKSTATION • DELETE SEGMENT • DELETE SEGMENT FROM WORKSTATION • SET SEGMENT PRIORITY • SET SEGMENT TRANSFORMATION • SET VISIBILITY 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.

(continued on next page)

Table 8–1 (Cont.) Surface Regeneration from Changes to Segments

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, as 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 DELETE SEGMENT or DELETE SEGMENT FROM WORKSTATION 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 (see Chapter 6), the workstation requires an implicit regeneration to affect this change.

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 they are not lost when you update the surface. For complete information as to changes that may require implicit regeneration on UPDATE WORKSTATION, or on REDRAW ALL SEGMENTS ON WORKSTATION, see Chapter 4.

8.4 Segment Attributes

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 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.

8.4.1 Detectability

The detectability segment attribute determines whether or not the segment can be chosen during pick input. Pick input is only available on OUTIN workstations. By default, DEC GKS segments are undetectable (GUNDET).

To pick a segment, it must be both detectable and visible (GVISI). 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, see SET DETECTABILITY in this chapter. For more information concerning pick input, see Chapter 9.

8.4.2 Highlighting

The highlighting 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 (GNORML).

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 the NDC points of the primitives stored in the segment. For more information concerning highlighting, see SET HIGHLIGHTING in this chapter.

8.4.3 Priority

The priority 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. A device supports either an infinite number of priorities (theoretically), or a specific number of priorities. If the device supports an infinite number 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 INQUIRE NUMBER OF SEGMENT PRIORITIES SUPPORTED.)

If the number of priorities supported is not 0, 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 example, 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.

8.4.4 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.

The first decision you must make when working with segment transformations is whether to specify your fixed point as a WC or NDC point. If you want to transform portions of the segment according to the *current* normalization transformation mapping, specify WC points. DEC GKS maps the specified WC point to NDC space 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 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 point around which the other points in the segment rotate. If the rotation is three-dimensional, the fixed point is the origin for the X, Y, and Z axes of rotation.

If you decide to shift the segment, you need to establish a **translation vector**. The translation vector is expressed by 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 system. If you do not wish to translate the segment's position, you can specify the value 0.0 for all components of the translation vector. The two-dimensional translation vector has X and Y components only. The three-dimensional translation vector has X, Y, and Z components, and its use is analogous to the two-dimensional 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. For three dimensions, DEC GKS rotates the segment through the specified angle of rotation about the specified axis through the fixed point. For two dimensions, DEC GKS rotates the segment through the specified angle of rotation about the fixed axis. 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 all components of the scale factor. Values less than 1.0 decrease the segment size, and values greater than 1.0 increase the segment size. The two-dimensional scale factor has X and Y components only. The three-dimensional scale factor has X, Y, and Z components, and its use is analogous to the two-dimensional scale factor.

Once you have decided how to scale, rotate, and translate a segment, you must construct a **transformation matrix**. A transformation matrix is an array of real values. The two-dimensional transformation matrix has six elements; the three-dimensional transformation matrix has twelve elements. To assist you in the creation of a transformation matrix, DEC GKS provides the utility functions EVALUATE TRANSFORMATION MATRIX (3) and ACCUMULATE TRANSFORMATION MATRIX (3). The function EVALUATE TRANSFORMATION MATRIX has the following function syntax:

```
GEVTM ( X0, Y0, DX, DY, PHI, SCX, SCY, SW, MO )
```

After evaluating the first eight arguments, EVALUATE TRANSFORMATION MATRIX establishes the appropriate transformation matrix and writes the 6-element array of real numbers to the last argument *MO*. For detailed information concerning this function, see the function description in Chapter 7.

The function ACCUMULATE TRANSFORMATION MATRIX is identical to EVALUATE TRANSFORMATION MATRIX, except that its first read-only argument is another 6-element transformation matrix, as follows:

```
GACTM ( MI, X0, Y0, DX, DY, PHI, SCX, SCY, SW, MO )
```

If you have a previously constructed transformation matrix to which you want to add translation, shifting, and scaling values, you call ACCUMULATE TRANSFORMATION 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. For detailed information concerning this function, see the function description in Chapter 7.

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 SET SEGMENT TRANSFORMATION (3), 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 system. This current transformation remains in effect until you change it. Before copying 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 to see the change in the segment transformation. See Section 8.3 for more information concerning surface update.

See Example 7–2 for a program example on the effects of a segment transformation.

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 order other than scaling, then rotating, and finally translation. You can accomplish this task by calling ACCUMULATE TRANSFORMATION MATRIX (3) several times, performing one transformation at a time.

See Example 7–1 for a program example showing the cumulative effect of ACCUMULATE TRANSFORMATION MATRIX.

8.4.4.1 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 or volume (the current normalization viewport), and the pick identifier value (see Section 8.1.1).

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, or ([0,1] x [0,1] x [0,1]) as the clipping volume, for the generated primitive.

Consequently, when you translate a segment's position, 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.

During transformation, a segment's primitives may exceed the default normalization viewport, defined as ([0,1] x [0,1]) for two-dimensional transformations, and ([0,1] x [0,1] x [0,1]) for three-dimensional transformations. 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. The maximum workstation window is ([0,1] x [0,1]) for two-dimensional transformations on the NDC plane and ([0,1] x [0,1] x [0,1]) for three-dimensional transformations in NPC space.

See Example 7–3 for a program example on controlling clipping at the world viewport.

8.4.5 Visibility

The visibility segment attribute determines if the segment is visible on the workstation surface. By default, DEC GKS segments are visible (GVISI).

Visibility can be used to hide a segment from a user until the segment is needed. For example, segment visibility is a useful way to control the displaying of messages and menus, although MESSAGE and REQUEST CHOICE can perform the same task.

By default, the visibility segment attribute is set to (GVISI). Keep in mind that a segment must be both visible and detectable to pick that segment during pick input (see Chapter 9).

8.5 Segment Inquiries

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

INQUIRE CLIPPING (3)
INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES (3)
INQUIRE LEVEL OF GKS
INQUIRE NAME OF OPEN SEGMENT
INQUIRE OPERATING STATE VALUE
INQUIRE PICK DEVICE STATE
INQUIRE SEGMENT ATTRIBUTES (3)
INQUIRE SET OF ACTIVE WORKSTATION
INQUIRE SET OF ASSOCIATED WORKSTATIONS
INQUIRE SET OF OPEN WORKSTATIONS
INQUIRE SET OF SEGMENT NAMES IN USE
INQUIRE SET OF SEGMENT NAMES ON WORKSTATION
INQUIRE WORKSTATION CATEGORY
INQUIRE WORKSTATION CONNECTION AND TYPE
INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES
INQUIRE WORKSTATION MAXIMUM NUMBERS
INQUIRE WORKSTATION STATE

For more information concerning device-independent programming, see the *DEC GKS User's Guide*.

8.6 Function Descriptions

This section describes the DEC GKS segment functions in detail.

ASSOCIATE SEGMENT WITH WORKSTATION

Operating States

WSOP, WSAC

Syntax

GASGWK (WKID, SGNA)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
SGNA	Integer	Read	Segment name that identifies a segment stored on a WISS workstation.

Description

The ASSOCIATE SEGMENT WITH WORKSTATION function takes a segment stored in workstation-independent segment storage (WISS), and stores the segment on the specified workstation. If the segment is not stored in WISS, DEC GKS generates an error.

Clipping volumes, clipping indicators, and view indexes are stored unchanged. This function cannot be invoked when a segment is open.

If the segment is already associated with the specified workstation, this function has no effect.

See Also

COPY SEGMENT TO WORKSTATION
INSERT SEGMENT
Example 8–1 for a program example using the ASSOCIATE SEGMENT WITH WORKSTATION function

CLOSE SEGMENT

Operating States

SGOP

Syntax

GCLSG ( )

Description

The CLOSE SEGMENT function 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 SGOP (segment open) to WSAC (at least one workstation active).

See Also

CREATE SEGMENT
Example 8–1 for a program example using the CLOSE SEGMENT function

COPY SEGMENT TO WORKSTATION

Operating States

WSOP, WSAC

Syntax

GCSGWK (WKID, SGNA)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
SGNA	Integer	Read	Segment name. This segment must be stored on a WISS workstation.

Description

The COPY SEGMENT TO WORKSTATION function copies the output primitives from a segment stored on the WISS workstation to the specified workstation.

As part of the copy operation, the output primitives are transformed by the segment transformation stored with the segment, clipped by the clipping volume or rectangle stored with the primitive, entered into the two- or three-dimensional transformation pipeline before the normalization clipping step, and are finally transformed by the workstation transformation and output. See the *DEC GKS User's Guide* for more information.

Primitives are clipped by the clipping volume or rectangle only when the clipping indicator stored with the primitive is set to CLIP.

See Also

ASSOCIATE SEGMENT WITH WORKSTATION
INSERT SEGMENT
Example 8–1 for a program example using the COPY SEGMENT TO WORKSTATION function

CREATE SEGMENT

Operating States

WSAC

Syntax

GCRSG (SGNA)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name. You cannot use the same name for two different segments. You must use positive integers as segment names.

Description

The CREATE SEGMENT function opens a segment on all active workstations.

When you call CREATE SEGMENT, the DEC GKS operating state is changed from at least one workstation active (WSAC) to segment open (SGOP).

For every active workstation, the name of the segment is added to the list of segments stored on that workstation.

The segment state list is set to the initial state of segment visible, undetectable, and not highlighted. The segment priority is set to 0, and the segment transformation is set to the identity transformation.

All subsequent output primitives will be added to the segment until the next CLOSE SEGMENT function is performed.

Only one segment can be open at a time.

See Also

CLOSE SEGMENT
DELETE SEGMENT
RENAME SEGMENT
Example 8–1 for a program example using the CREATE SEGMENT function

DELETE SEGMENT

Operating States

WSOP, WSAC, SGOP

Syntax

GDSG (SGNA)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name

Description

The DELETE SEGMENT function deletes the specified segment from all workstations storing that segment. Using this function, you can delete any defined segment, but you cannot delete an open segment.

Calling this function deletes the specified segment's state list.

See Also

DELETE SEGMENT FROM WORKSTATION

DELETE SEGMENT FROM WORKSTATION

Operating States

WSOP, WSAC, SGOP

Syntax

GDSGWK (WKID, SGNA)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
SGNA	Integer	Read	Segment name

Description

The DELETE SEGMENT FROM WORKSTATION function deletes the segment from the specified workstation. Using this function, 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 calling the function DELETE SEGMENT.

See Also

DELETE SEGMENT

INSERT SEGMENT

Operating States

WSAC, SGOP

Syntax

GINSG (SGNA, M)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name
M(2,3)	Array of reals	Read	Insertion transformation matrix: $\begin{bmatrix} M(1,1) M(1,2) M(1,3) \\ M(2,1) M(2,2) M(2,3) \end{bmatrix}$

Description

The INSERT SEGMENT function takes the specified segment stored in a WISS workstation and, for each active workstation, copies the primitives in the specified segment to either the open segment or into the stream of primitives outside segments. The primitives in the segment are transformed twice: once according to the segment's current transformation, and once according to the transformation matrix specified in the call to this function (the effect of the two transformations is cumulative).

For each active workstation, the primitives contained in the specified segment are copied to the workstation. The primitives are added to the open segment if the GKS state is SGOP (segment open). They are added to the stream of primitives outside of segments if the GKS state is WSAC (at least one workstation active).

All segments have a segment transformation attribute. The segment transformation is stored with the segment when the segment is created.

Output primitives stored in segments have a clipping volume and clipping indicator stored with the primitive when the primitive is created. The stored clipping rectangle and clipping indicator are taken from the GKS state list. The primitives contained in the specified segment are transformed first by the segment transformation of the specified segment. Then they are transformed by the specified 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 calculation, in that order. You can formulate an insertion transformation using either EVALUATE TRANSFORMATION MATRIX or ACCUMULATE TRANSFORMATION MATRIX.

The following equation shows the specified insertion transformation matrix. It also shows the effect of applying the specified insertion transformation matrix to a coordinate that already has been transformed by the segment transformation.

The insert transformation and the segment transformation (conceptually) take place in NDC space.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The original coordinates are (x, y) and the transformed coordinates are (x′, y′). Both of these coordinates are NDC points. The values M_{13} and M_{23} of the insertion transformation matrix are also specified in NDC units. All other matrix elements are unitless.

As part of the copy operation, all segment attributes (except the segment transformation attribute) are ignored. The ignored attributes are segment visibility, highlighting, detectability, and priority.

Also as part of the copy operation, the current settings of the attributes in the GKS state list replace all the clipping indicators and clipping rectangles in the specified segment.

During the copy operation, all primitives in the specified segment retain the values of their corresponding primitive attributes that were assigned to them when they were created.

INSERT SEGMENT does not affect the current settings of the primitive attributes in the GKS state list.

See Also

ACCUMULATE TRANSFORMATION MATRIX
ASSOCIATE SEGMENT WITH WORKSTATION
COPY SEGMENT TO WORKSTATION
EVALUATE TRANSFORMATION MATRIX
SET SEGMENT TRANSFORMATION
Example 8–2 for a program example using the INSERT SEGMENT function

INSERT SEGMENT 3

Operating States

WSAC, SGOP

Syntax

GINSG3 (SGNA, M)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name
M(3,4)	Array of reals	Read	Insertion transformation matrix: $\begin{bmatrix} M(1,1) & M(1,2) & M(1,3) & M(1,4) \\ M(2,1) & M(2,2) & M(2,3) & M(2,4) \\ M(3,1) & M(3,2) & M(3,3) & M(3,4) \end{bmatrix}$

Description

The INSERT SEGMENT 3 function takes the specified segment stored in a WISS workstation and, for each active workstation, copies the primitives in the specified segment to either the open segment or into the stream of primitives outside segments. The primitives in the segment are transformed twice: once according to the segment's current transformation, and once according to the transformation matrix specified in the call to this function (the effect of the two transformations is cumulative).

For each active workstation, the primitives contained in the specified segment are copied to the workstation. The primitives are added to the open segment if the GKS state is SGOP (segment open). They are added to the stream of primitives outside of segments if the GKS state is WSAC (at least one workstation active).

All segments have a segment transformation attribute. The segment transformation is stored with the segment when the segment is created.

Output primitives stored in segments have a clipping volume and clipping indicator stored with the primitive when the primitive is created. The stored clipping volume and clipping indicator are taken from the GKS state list. The primitives contained in the specified segment are transformed first by the segment transformation of the specified segment. Then they are transformed by the specified 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 calculation, in that order. You can formulate an insertion transformation using either EVALUATE TRANSFORMATION MATRIX 3 or ACCUMULATE TRANSFORMATION MATRIX 3.

The following equation shows the specified insertion transformation matrix. It also shows the effect of applying the specified insertion transformation matrix to a coordinate that already has been transformed by the segment transformation. The insert transformation and the segment transformation (conceptually) take place in NDC space.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The original coordinates are (x, y, z) and the transformed coordinates are (x′, y′, z′). Both of these coordinates are NDC points. The values M_{14}, M_{24}, and M_{34} of the insertion transformation matrix are also specified in NDC units. All other matrix elements are unitless.

As part of the copy operation, all segment attributes (except the segment transformation attribute) are ignored. The ignored attributes are segment visibility, highlighting, detectability, and priority.

Also as part of the copy operation, the current settings of the attributes in the GKS state list replace all the clipping indicators and clipping rectangles in the specified segment.

During the copy operation, all primitives in the specified segment retain the values of their corresponding primitive attributes that were assigned to them when they were created.

INSERT SEGMENT 3 does not affect the current settings of the primitive attributes in the GKS state list.

See Also

ACCUMULATE TRANSFORMATION MATRIX 3
ASSOCIATE SEGMENT WITH WORKSTATION
COPY SEGMENT TO WORKSTATION
EVALUATE TRANSFORMATION MATRIX 3
SET SEGMENT TRANSFORMATION 3
Example 8–2 for a program example using the INSERT SEGMENT function

RENAME SEGMENT

Operating States

WSOP, WSAC, SGOP

Syntax

GRENSG (OLD, NEW)

Argument	Data Type	Access	Description
OLD	Integer	Read	Current segment name. After the call to this function, this identifier can be used to name another segment.
NEW	Integer	Read	New segment name.

Description

The RENAME SEGMENT function changes the segment name from its current name to a new name. After you have renamed a segment using this function, you can reuse the old segment name.

SET DETECTABILITY

Operating States

WSOP, WSAC, SGOP

Syntax

GSDTEC (SGNA, DET)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name
DET	Integer (constant)	Read	Flag that determines whether the user can pick the segment

Constants

Defined Argument	Constant	Description
DET	GUNDET	Segment cannot be picked. This is the default value.
	GDETEC	Segment, if visible, can be picked.

Description

The SET DETECTABILITY function controls the segment attribute that determines whether the specified segment can be chosen during pick input. A segment must be both detectable and visible to be picked. The detectability segment attribute is described in more detail in Section 8.4.1.

See Also

SET VISIBILITY
Example 9–2 for a program example using the SET DETECTABILITY function

SET HIGHLIGHTING

Operating States

WSOP, WSAC, SGOP

Syntax

GSHLIT (SGNA, HIL)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name
HIL	Integer (constant)	Read	Highlighting flag

Constants

Defined Argument	Constant	Description
HIL	GNORML	DEC GKS does not highlight the segment. This is the default value.
	GHILIT	DEC GKS highlights the segment, if visible.

Description

The SET HIGHLIGHTING function controls the segment attribute that determines whether the specified segment is highlighted.

If you use this function to highlight a segment on a VT241™ terminal, 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. For more information on segment highlighting, see Section 8.4.2.

See Also

Example 8–3 for a program example using the SET HIGHLIGHTING function

SET SEGMENT PRIORITY

Operating States

WSOP, WSAC, SGOP

Syntax

GSSGP (SGNA, PRIOR)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name.
PRIOR	Real	Read	Segment priority, between 0.0 and 1.0. The initial segment priority value is 0.0.

Description

The SET SEGMENT PRIORITY function sets the segment attribute that 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 0.0 to 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. A device supports either an infinite number of priorities (theoretically) or a specific number of priorities. For more information on segment priority, see Section 8.4.3.

See Also

GET PICK
REQUEST PICK
SAMPLE PICK

SET SEGMENT TRANSFORMATION

Operating States

WSOP, WSAC, SGOP

Syntax

GSSGT (SGNA, M)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name
M(2,3)	Array of reals	Read	Transformation matrix: $\begin{bmatrix} M(1,1) M(1,2) M(1,3) \\ M(2,1) M(2,2) M(2,3) \end{bmatrix}$

Description

The SET SEGMENT TRANSFORMATION function specifies the segment transformation that is stored with a specified segment.

All segments have a segment transformation attribute. The segment transformation is stored with the segment when the segment is created. At the time a segment is created, its segment transformation is set to the identity transformation.

SET SEGMENT TRANSFORMATION changes the segment transformation of the specified segment. When a segment is displayed, its primitives are transformed by the specified transformation matrix according to the following equation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The original coordinates are (x, y); the transformed coordinates are (x′, y′). Both of these coordinates are NDC points. The values M_{13} and M_{23} of the segment transformation matrix are also specified in NDC units. All other matrix elements are unitless.

The functions EVALUATE TRANSFORMATION MATRIX and ACCUMULATE TRANSFORMATION MATRIX facilitate the construction of matrixes that can be specified as segment transformation matrixes.

The segment transformation can be reset by calling this function with the identity matrix. Segment transformations do not affect locator and stroke input coordinates. Segment transformation is described in more detail in Section 8.4.4.

See Also

ACCUMULATE TRANSFORMATION MATRIX
EVALUATE TRANSFORMATION MATRIX
Example 7–1 for a program example using the SET SEGMENT TRANSFORMATION function

SET SEGMENT TRANSFORMATION 3

Operating States

WSOP, WSAC, SGOP

Syntax

GSSGT3 (SGNA, M)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name
M(3,4)	Array of reals	Read	Transformation matrix:

$$\begin{bmatrix} M(1,1)\,M(1,2)\,M(1,3)\,M(1,4) \\ M(2,1)\,M(2,2)\,M(2,3)\,M(2,4) \\ M(2,1)\,M(2,2)\,M(2,3)\,M(3,4) \end{bmatrix}$$

Description

The SET SEGMENT TRANSFORMATION 3 function specifies the segment transformation that is stored in a specified segment.

All segments have a segment transformation attribute. The segment transformation is stored with the segment when the segment is created. At the time a segment is created, its segment transformation is set to the identity transformation.

SET SEGMENT TRANSFORMATION 3 changes the segment transformation of the specified segment. When a segment is displayed, its primitives are transformed by the specified transformation matrix according to the following equation:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The original coordinates are (x, y, z); the transformed coordinates are (x′, y′, z′). Both of these coordinates are NDC points. The values M_{14}, M_{24}, and M_{34} of the segment transformation matrix are also specified in NDC units. All other matrix elements are unitless.

The functions EVALUATE TRANSFORMATION MATRIX 3 and ACCUMULATE TRANSFORMATION MATRIX 3 facilitate the construction of matrixes that can be specified as segment transformation matrixes.

The segment transformation can be reset by calling this function with the identity matrix. Segment transformations do not affect locator and stroke input coordinates. Segment transformation is described in more detail in Section 8.4.4.

See Also

ACCUMULATE TRANSFORMATION MATRIX 3
EVALUATE TRANSFORMATION MATRIX 3
Example 7–1 for a program example using the SET SEGMENT TRANSFORMATION function

SET VISIBILITY

Operating States

WSOP, WSAC, SGOP

Syntax

GSVIS (SGNA, VIS)

Argument	Data Type	Access	Description
SGNA	Integer	Read	Segment name
VIS	Integer (constant)	Read	Visibility flag

Constants

Defined Argument	Constant	Description
VIS	GINVIS	DEC GKS does not show the segment on the workstation surface.
	GVISI	DEC GKS shows the segment on the workstation surface. This is the default value.

Description

The SET VISIBILITY function sets the segment attribute that determines whether the specified segment is visible on the workstation surface. A segment must be both visible and detectable to be picked.

Depending on the capabilities of the device, and whether or not the specified segment overlaps other segments, you may need to call either UPDATE WORKSTATION or REDRAW ALL SEGMENTS ON WORKSTATION to update the picture on the surface of the workstation. For more information, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*. The visibility segment attribute is described in more detail in Section 8.4.5.

See Also

REDRAW ALL SEGMENTS ON WORKSTATION
SET DETECTABILITY
UPDATE WORKSTATION

8.7 Program Examples

Example 8–1 illustrates the use of the ASSOCIATE SEGMENT WITH WORKSTATION function.

Example 8–1 Comparing ASSOCIATE SEGMENT WITH WORKSTATION and COPY SEGMENT TO WORKSTATION

```
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 to show the ASSOCIATE SEGMENT WITH
C  WORKSTATION and COPY SEGMENT TO WORKSTATION functions.
C
C  NOTE: To keep the example concise, no error checking is performed.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      REAL TIME_OUT
      INTEGER DEV_NUM
      INTEGER  HOUSE
      INTEGER IN_CLASS
      REAL  LARGER
      INTEGER  LOWER_LEFT_CORNER
      INTEGER NUM_POINTS
      REAL PX (9)
      REAL  PY (9)
      INTEGER TITLE
      INTEGER UPPER_LEFT_CORNER
      INTEGER WISS
      REAL WORLD_X
      REAL WORLD_Y
      INTEGER   WS_ID

C  Open GKS and the workstation environments.

      WISS = 2
      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GOPWK (WISS, GCONID,  GWSWIS)

C  By activating only the WISS workstation, GKS stores created
C  segments only on the WISS workstation.  The screen remains clear.

      CALL GACWK (WISS)

C  Set the viewport limits for the specified normalization
C  transformations.

      LOWER_LEFT_CORNER = 1
      UPPER_LEFT_CORNER = 2

      CALL GSVP (LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5)
      CALL GSVP (UPPER_LEFT_CORNER, 0.0, 0.5, 0.5, 1.0)
```

(continued on next page)

Example 8–1 (Cont.) Comparing ASSOCIATE SEGMENT WITH WORKSTATION and COPY SEGMENT TO WORKSTATION

```
C  Create two segments and store them on the WISS workstation.

      HOUSE = 1
      NUM_POINTS = 9
      PX(1) = .4
      PY(1) = .1
      PX(2) = .1
      PY(2) = .1
      PX(3) = .1
      PY(3) = .7
      PX(4) = .4
      PY(4) = .7
      PX(5) = .25
      PY(5) = .9
      PX(6) = .1
      PY(6) = .7
      PX(7) = .4
      PY(7) = .1
      PX(8) = .4
      PY(8) = .7
      PX(9) = .1
      PY(9) = .1
      TITLE = 2


      CALL GCRSG (HOUSE)
      CALL GSELNT (LOWER_LEFT_CORNER)
      CALL GPL (NUM_POINTS, PX, PY)
      CALL GCLSG ()

      CALL GCRSG (TITLE)
      CALL GSELNT (UPPER_LEFT_CORNER)

      LARGER = .03
      WORLD_X = .1
      WORLD_Y = .5

      CALL GSCHH (LARGER)
      CALL GTX (WORLD_X, WORLD_Y, 'Associated segment.')
      CALL GCLSG ()

C  Activate the WS_ID workstation.

      CALL GACWK (WS_ID)

C  By associating the segment containing text, the workstation stores
C  the segment. By copying the segment containing the house, GKS draws
C  the primitives to the display screen, but the workstation does not
C  store the segment.

      CALL GASGWK (WS_ID, TITLE)
      CALL GCSGWK (WS_ID, HOUSE)

C  Release the deferred output.  Wait 5 seconds.

      TIME_OUT = 5.00

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)
```

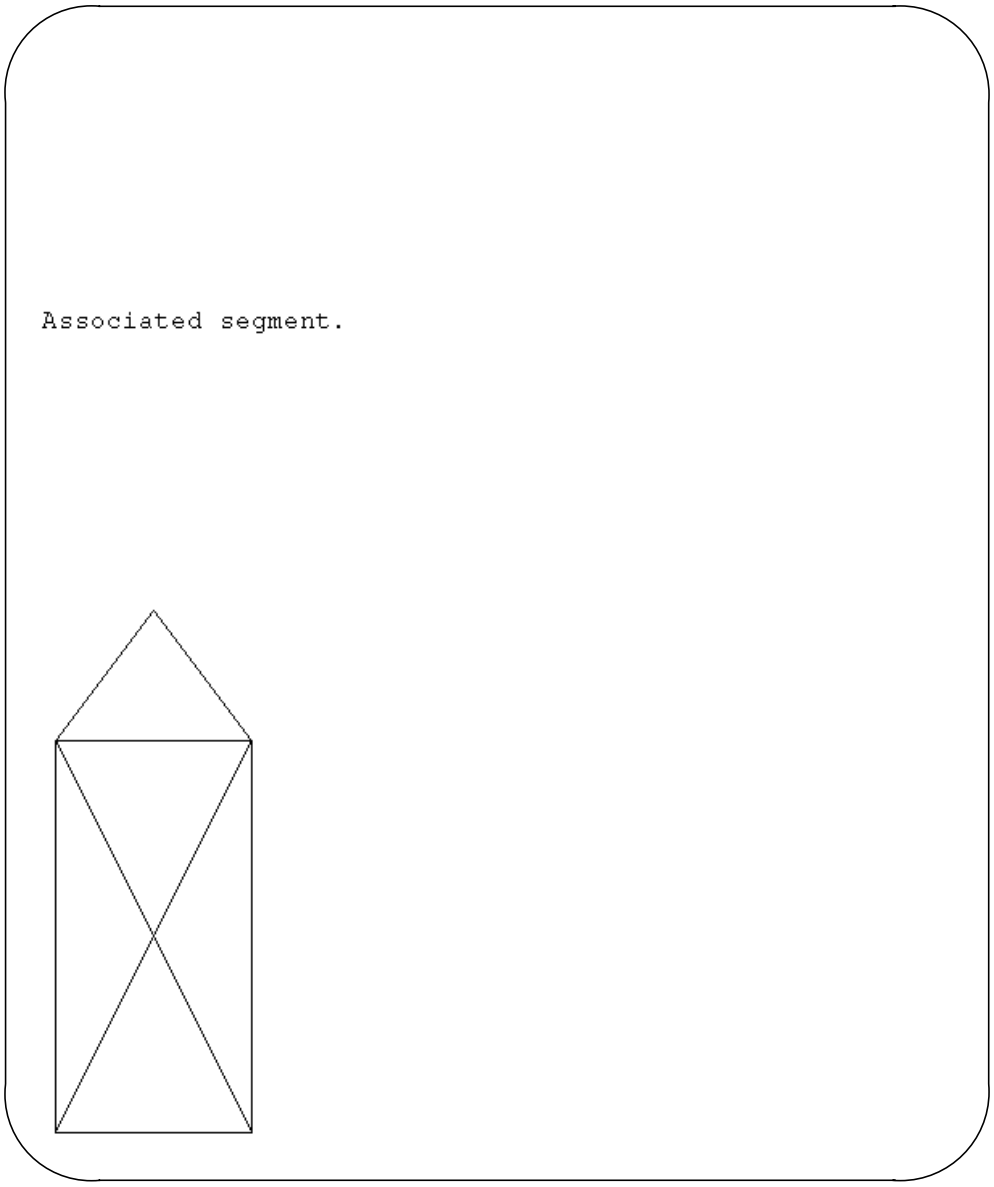
(continued on next page)

Example 8–1 (Cont.) Comparing ASSOCIATE SEGMENT WITH WORKSTATION and COPY SEGMENT TO WORKSTATION

```
C  When you redraw the segments, you force an update to the screen
C  and eliminate primitives not contained in segments.  The house
C  disappears and the text remains on the display screen because it
C  was stored as a segment.

      CALL GRSGWK (WS_ID)

C  Deactivate and close the workstation environments and GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GDAWK (WISS)
      CALL GCLWK (WISS)
      CALL GCLKS ()
      END
```

Figure 8–1 shows the two segments (the house and the line of text) on the display surface.

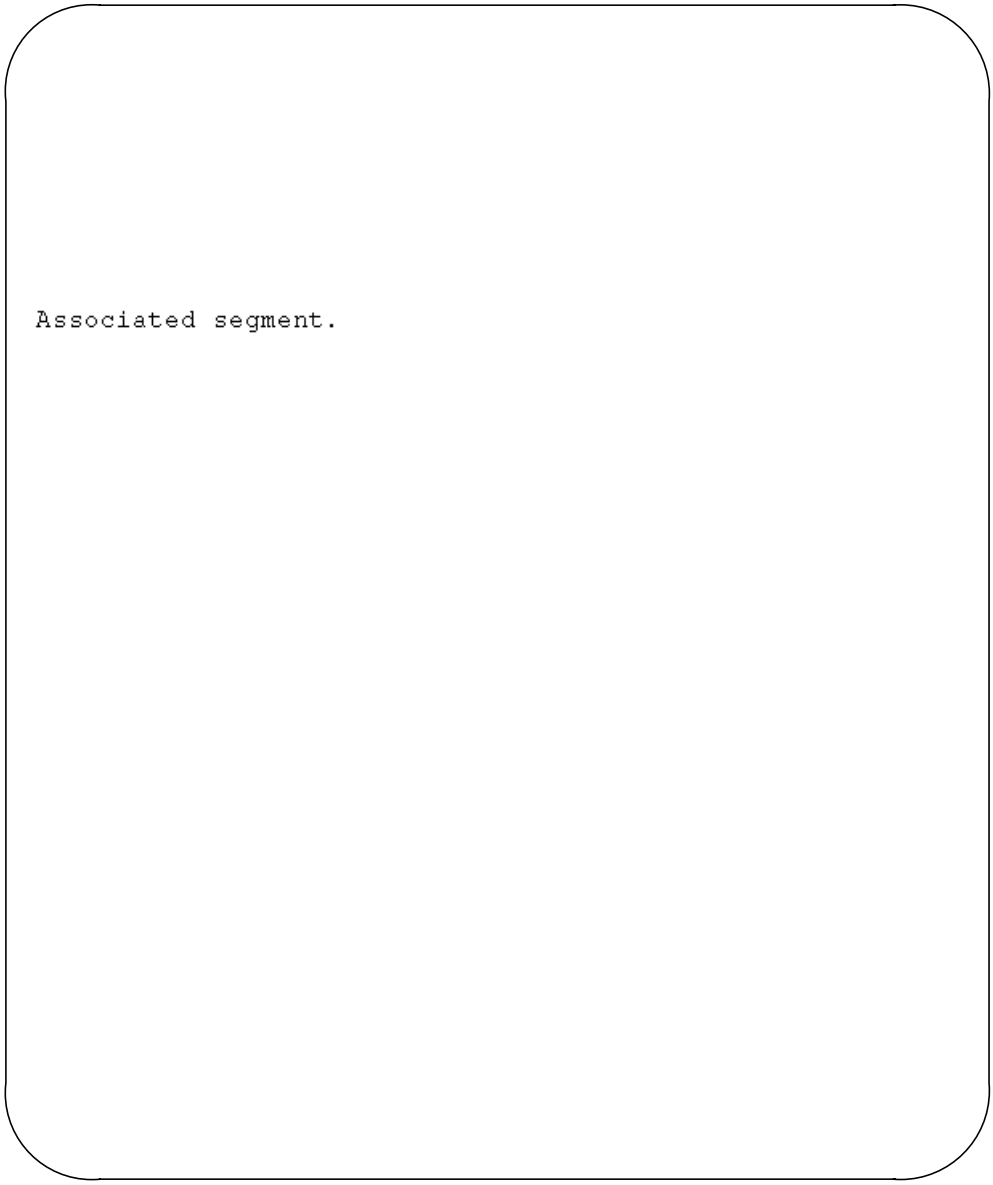
Figure 8–1 Output with Two Segments



ZK–4017A–GE

Figure 8–2 shows only the text, which was stored as a segment.

Figure 8–2 Output with Associated Segment



ZK–4018A–GE

Example 8–2 illustrates the use of the INSERT SEGMENT function.

Example 8–2 Inserting a Segment's Primitives into Another Segment

```
C  This program illustrates the use of the function INSERT SEGMENT.
C  It draws a house in the lower left corner of the screen and then
C  inserts that house into other segments.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER DEV_NUM
      INTEGER HOUSE_1
      INTEGER  HOUSE_2
      INTEGER IN_CLASS
      INTEGER LOWER_LEFT_CORNER
      REAL  NO_CHANGE
      REAL NULL
      INTEGER NUM_POINTS
      REAL  PX (9)
      REAL PY (9)
      REAL RIGHT
      REAL TIME_OUT
      REAL  UP
      INTEGER   WISS
      INTEGER WS_ID
      REAL XFORM_MATRIX (6)

C  Open and activate GKS and the workstation environments.

      WISS = 2
      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GOPWK (WISS, GCONID, GWSWIS)
      CALL GACWK (WS_ID)
      CALL GACWK (WISS)

C  Set the viewport limits for the normalization transformation.
C  The normalization window is established to be the lower left
C  corner of NDC space.

      LOWER_LEFT_CORNER = 1

      CALL GSVP (LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5)
```

(continued on next page)

Example 8–2 (Cont.) Inserting a Segment's Primitives into Another Segment

```
C  Create a segment in the lower left corner of the surface.

      HOUSE_1 = 1
      NUM_POINTS = 9
      PX(1) = .4
      PY(1) = .1
      PX(2) = .1
      PY(2) = .1
      PX(3) = .1
      PY(3) = .7
      PX(4) = .4
      PY(4) = .7
      PX(5) = .25
      PY(5) = .9
      PX(6) = .1
      PY(6) = .7
      PX(7) = .4
      PY(7) = .1
      PX(8) = .4
      PY(8) = .7
      PX(9) = .1
      PY(9) = .1

      CALL GCRSG (HOUSE_1)
      CALL GSELNT (LOWER_LEFT_CORNER)
      CALL GPL (NUM_POINTS, PX, PY)
      CALL GCLSG ()

C  Deactivate WISS so no other segments are stored there.

      CALL GDAWK (WISS)

C  Turn off the clipping so the transformed houses are visible.

      CALL GSCLIP (GNCLIP)

C  Change the matrix value.

      NO_CHANGE = 1.0
      NULL = 0.0
      RIGHT = 0.5

      CALL GEVTM (NULL, NULL, RIGHT, NULL, NULL, NO_CHANGE,
     *NO_CHANGE, GNDC, XFORM_MATRIX)

C  Create a segment in the lower right corner by inserting
C  the primitives for HOUSE_1 into HOUSE_2.

      HOUSE_2 = 2

      CALL GCRSG (HOUSE_2)
      CALL GINSG (HOUSE_1, XFORM_MATRIX)
      CALL GCLSG ()

C  Using EVALUATE TRANSFORMATION MATRIX, you can create the transformation
C  matrix that you need to pass to INSERT SEGMENT as an argument.  This
C  matrix specifies a position translation of 0.5 NDC points to the right.
C  When this matrix is passed to INSERT SEGMENT while a segment is open,
C  the house's primitives are transformed and made a part of the open
C  segment.

      UP = 0.5

      CALL GEVTM (NULL, NULL, NULL, UP, NULL, NO_CHANGE,
     *NO_CHANGE, GNDC, XFORM_MATRIX)
```

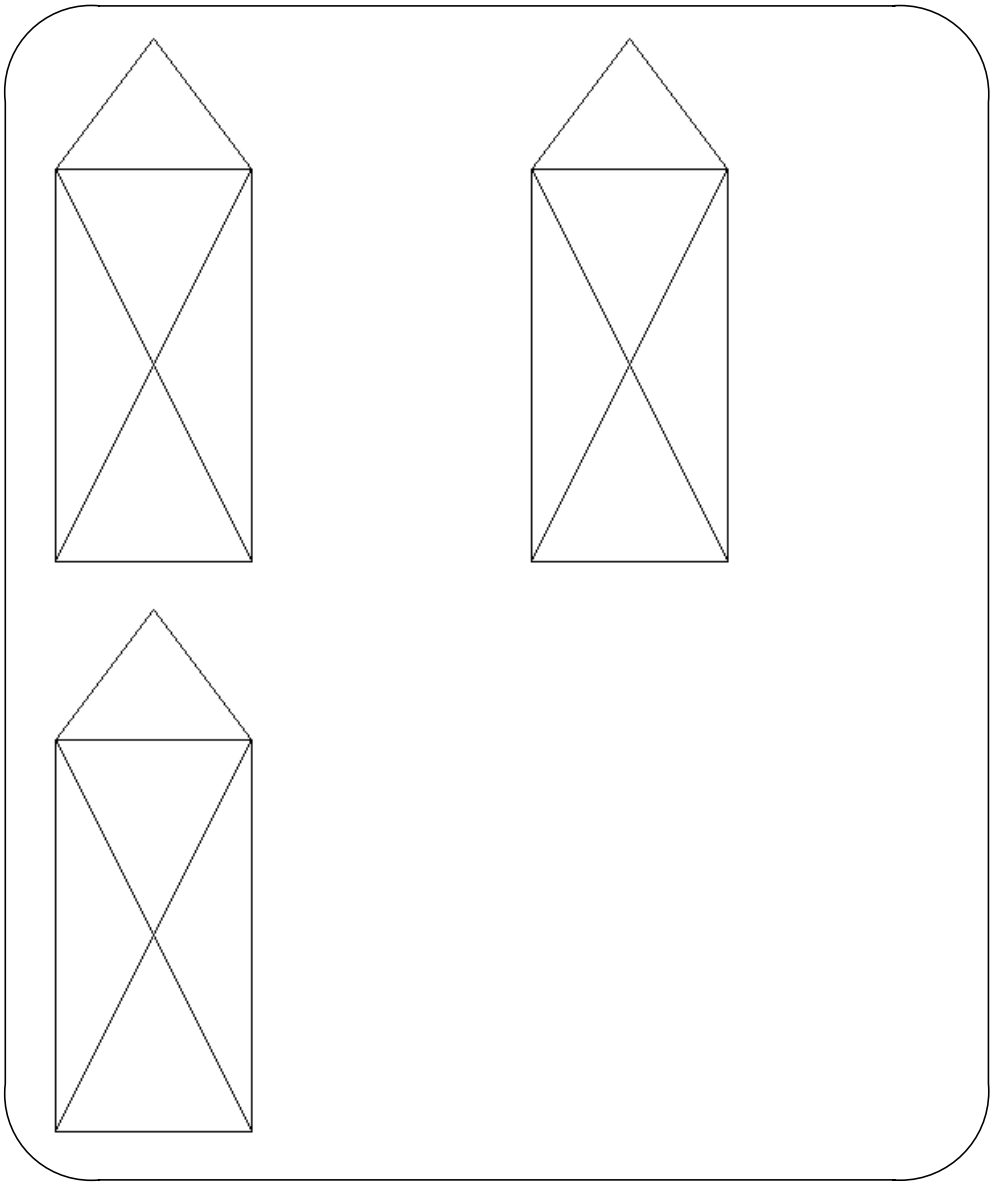
(continued on next page)

Example 8–2 (Cont.) Inserting a Segment's Primitives into Another Segment

```
C  Insert the primitives in the upper left corner using INSERT SEGMENT.
C  Inserting segments when the GKS operating state is GWSAC causes
C  the output primitives to be written to the workstation surface, but
C  the primitives are not stored in a segment.  These segment primitives
C  are translated 0.5 NDC points in an upwards direction.

      CALL GINSG (HOUSE_1, XFORM_MATRIX)

C  Change the matrix value.

      CALL GEVTM (NULL, NULL, RIGHT, UP, NULL, NO_CHANGE,
     *NO_CHANGE, GNDC, XFORM_MATRIX )

C  Insert the primitives in the upper right corner using INSERT SEGMENT.

      CALL GINSG (HOUSE_1, XFORM_MATRIX)

C  Release the deferred output. The display pauses for 5 seconds.

      TIME_OUT = 5.00

      CALL GUWK (WS_ID, GPOSTP)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  The call to REDRAW ALL SEGMENTS ON WORKSTATION redraws all segments
C  and deletes all primitives outside of segments.

      CALL GRSGWK (WS_ID)

C  Deactivate and close the workstation environments and GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WISS)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 8–3 shows the original segment, drawn in the lower left corner, inserted into the upper right and left corners of the display surface.

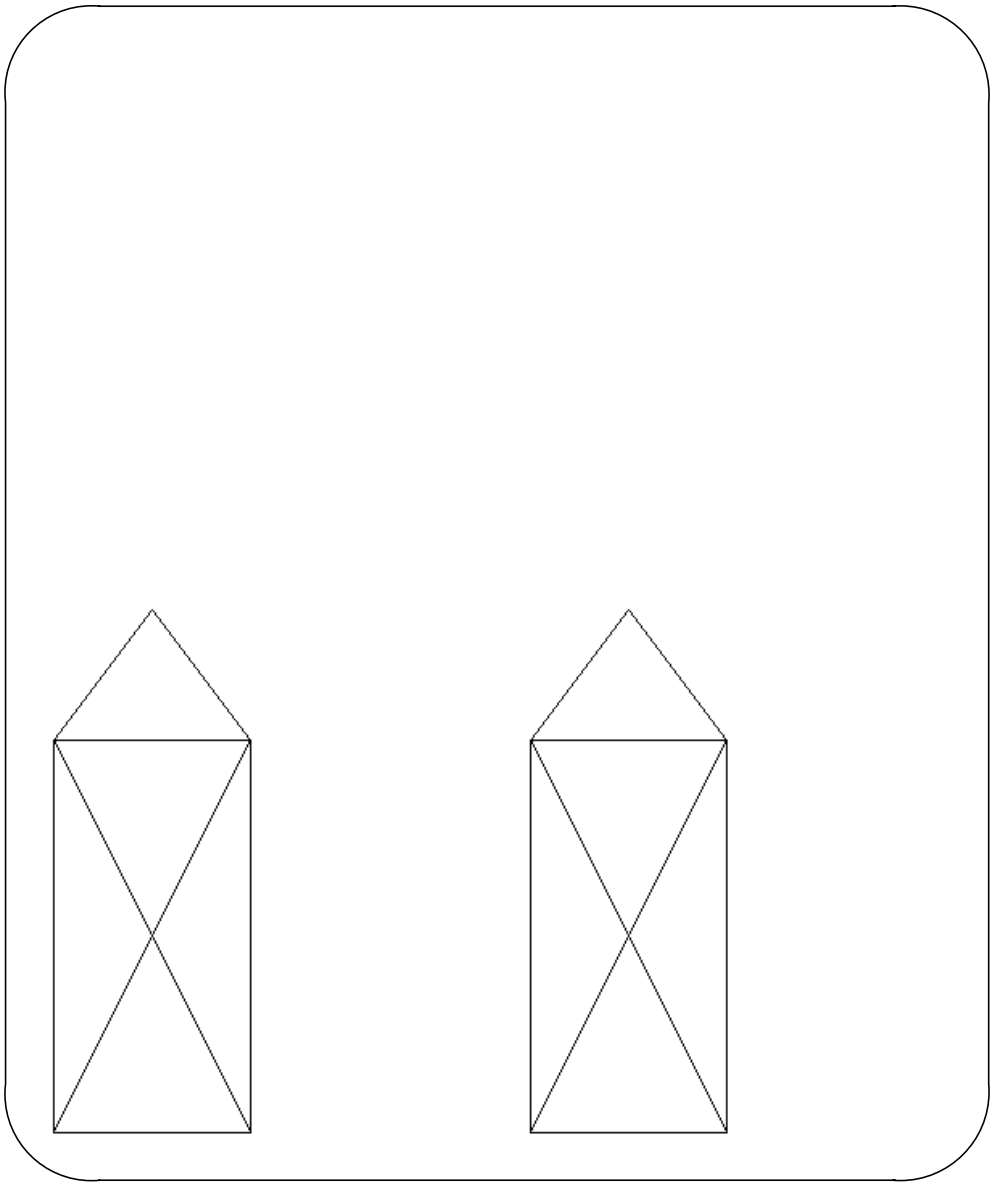
Figure 8–3 Output of Original and Inserted Segments



ZK–4023A–GE

Figure 8–4 shows the redrawn segments. The houses not stored in segments have been deleted.

Figure 8–4 Output of Redrawn Segments



ZK–4024A–GE

Example 8–3 illustrates the use of the SET HIGHLIGHTING function.

Example 8–3 Highlighting a Segment

```
C  This program illustrates the SET HIGHLIGHTING function.
C  It draws a house in the lower left corner of the screen and
C  a highlighted house in the upper right corner.

      IMPLICIT NONE
      INCLUDE 'gks.f'

      INTEGER DEV_NUM
      INTEGER HOUSE_1
      INTEGER  HOUSE_2
      INTEGER IN_CLASS
      INTEGER LOWER_LEFT_CORNER
      INTEGER NUM_POINTS
      REAL  PX (9)
      REAL PY (9)
      REAL TIME_OUT
      INTEGER UPPER_RIGHT_CORNER
      INTEGER WS_ID

C  Open and activate GKS and the workstation environment.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)

C  Set the viewport limits for the normalization transformations.

      LOWER_LEFT_CORNER = 1
      UPPER_RIGHT_CORNER = 2

      CALL GSVP (LOWER_LEFT_CORNER, 0.0, 0.5, 0.0, 0.5)
      CALL GSVP (UPPER_RIGHT_CORNER, 0.5, 1.0, 0.5, 1.0)

C  Create a segment in the lower left corner of the surface.

      HOUSE_1 = 1
      NUM_POINTS = 9
      PX(1) = .4
      PY(1) = .1
      PX(2) = .1
      PY(2) = .1
      PX(3) = .1
      PY(3) = .7
      PX(4) = .4
      PY(4) = .7
      PX(5) = .25
      PY(5) = .9
      PX(6) = .1
      PY(6) = .7
      PX(7) = .4
      PY(7) = .1
      PX(8) = .4
      PY(8) = .7
      PX(9) = .1
      PY(9) = .1

      CALL GCRSG (HOUSE_1)
      CALL GSELNT (LOWER_LEFT_CORNER)
      CALL GPL (NUM_POINTS, PX, PY)
      CALL GCLSG ()
```

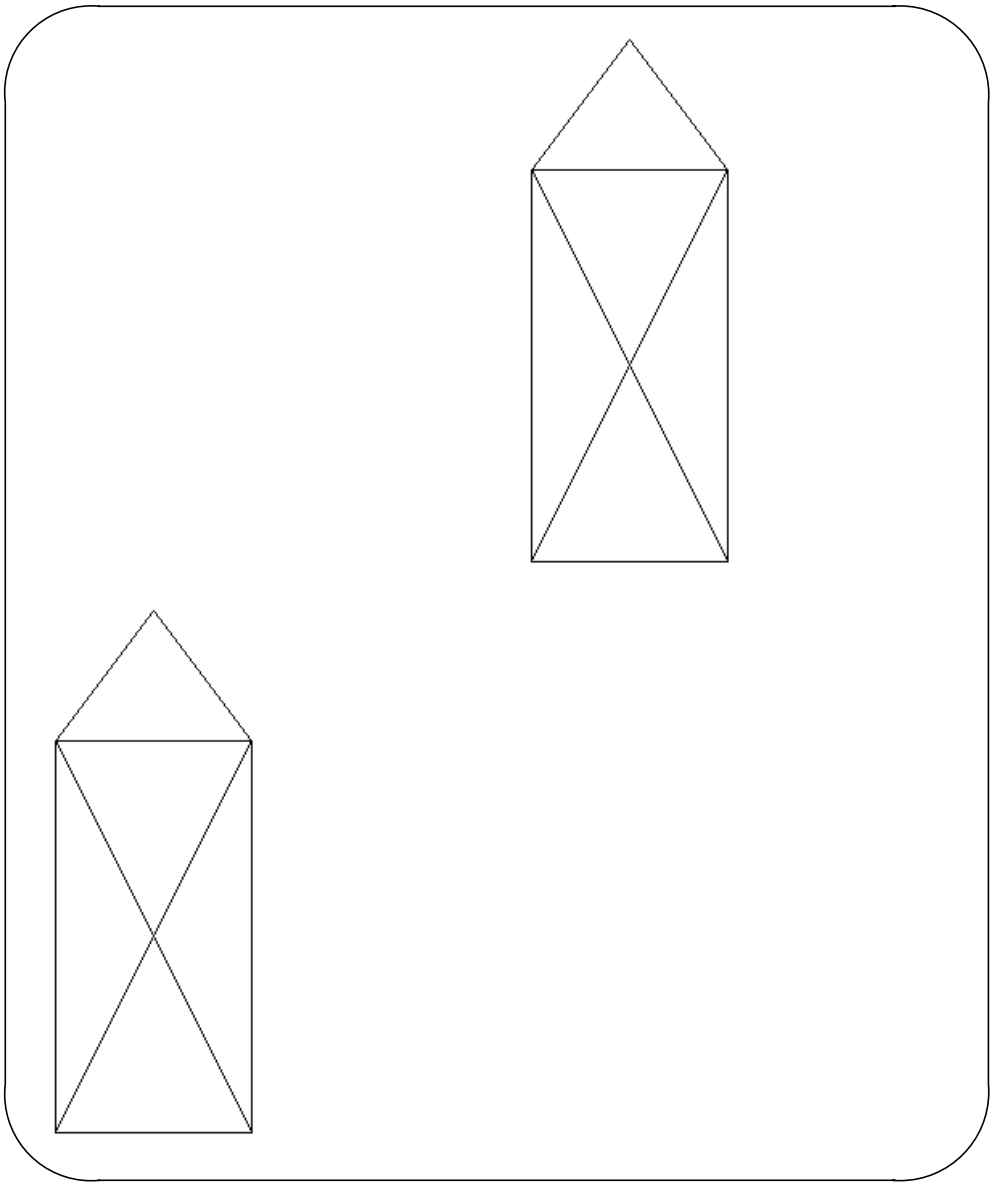
(continued on next page)

Example 8–3 (Cont.) Highlighting a Segment

```
C  Create a second segment in the upper right corner of the surface.

      HOUSE_2 = 2

      CALL GCRSG (HOUSE_2)
      CALL GSELNT (UPPER_RIGHT_CORNER)
      CALL GPL (NUM_POINTS, PX, PY)
      CALL GCLSG ()

C  Release the deferred output.  Wait 5 seconds.

      TIME_OUT = 5.00

      CALL GUWK (WS_ID, GPERFO)
      CALL GWAIT (TIME_OUT, WS_ID, IN_CLASS, DEV_NUM)

C  Highlight HOUSE_2.

      CALL GSHLIT (HOUSE_2, GHILIT)

C  Update the surface to initiate the change.

      CALL GUWK (WS_ID, GPERFO)

C  Deactivate and close the workstation environment and GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 8–5 illustrates the houses before highlighting occurs.

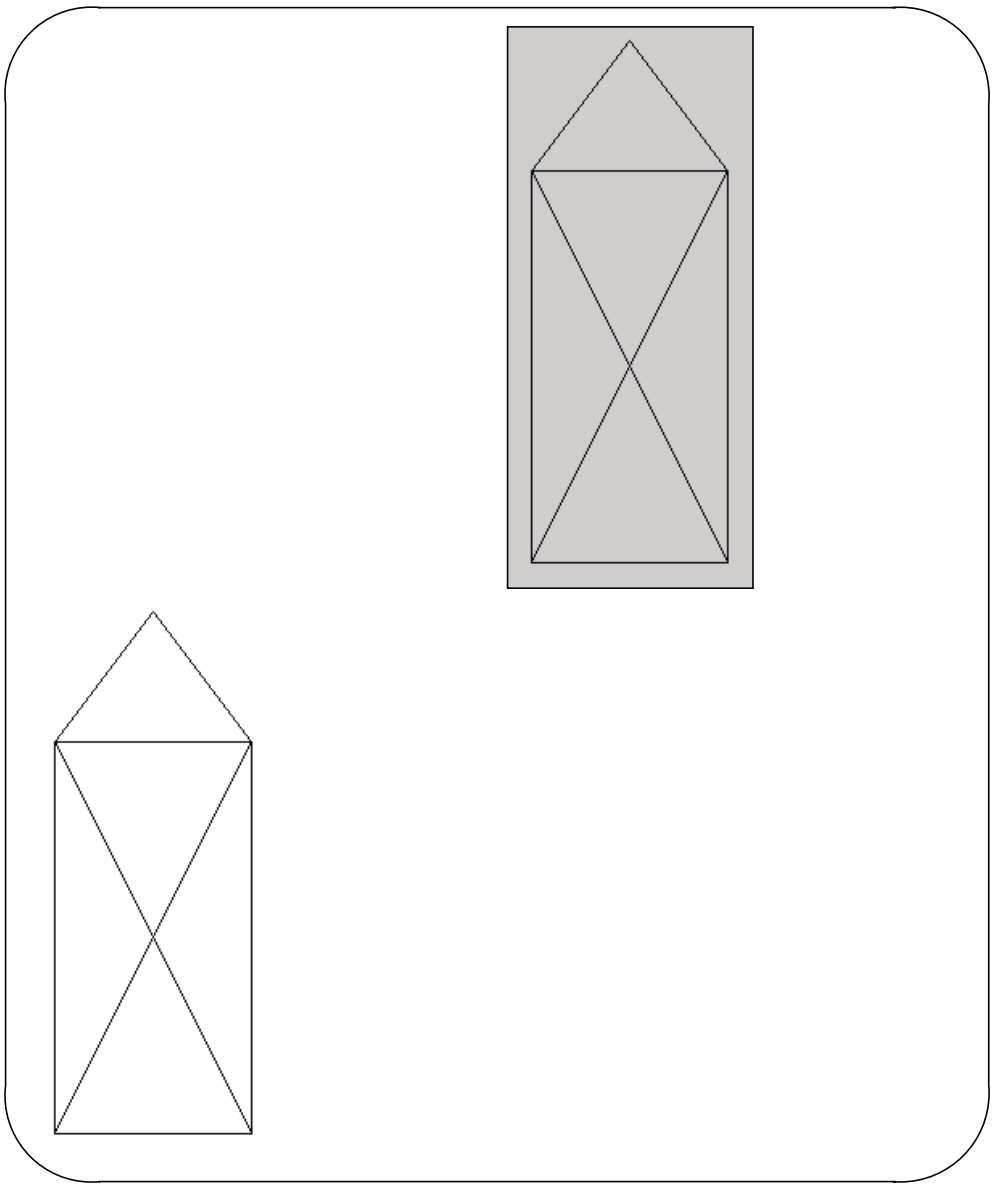
Figure 8–5 Output Prior to Highlighting



ZK–4025A–GE

Figure 8–6 shows the house in the upper right corner being highlighted.

Figure 8–6 Effects of SET HIGHLIGHTING



ZK–4026A–GE

Input Functions

Insert tabbed divider here. Then discard this sheet.

9
Input Functions

The DEC GKS input functions let an application accept input from a user. This chapter provides information about physical and logical input devices, prompt and echo types, and general information about input functions. Then it describes each DEC GKS input function.

9.1 Physical Input Devices

A **physical input device** provides input to an application. A keyboard, tablet, and mouse are examples of physical devices. A single application can use input from many physical input devices.

9.2 Logical Input Devices

Because many kinds of physical input devices exist, DEC GKS maintains device independence by using **logical input devices**. A logical input device acts as an intermediary between a physical input device and the application. That is, the user inputs data from the physical input device to the application via the logical input device. GKS lets a single open workstation have zero or more logical input devices active at the same time.

9.2.1 Identifying a Logical Input Device

A logical input device is identified by a workstation identifier, an input class, and a device number. The **workstation identifier** identifies an open workstation that belongs to the category INPUT or OUTIN. The logical input device is part of the workstation. How DEC GKS implements the logical input device depends on what physical input devices the workstation is using.

The **input class** determines the type of logical input value the logical input device returns to the application. A logical input device belongs to one of six input classes. For example, a locator-class logical input device returns cursor location values. You determine the input class when you activate the logical input device. (See Section 9.2.3 for information about activating a logical input device and Section 9.2.6 for a detailed description of input classes.)

The **device number** distinguishes one logical input device from another of the same input class on the same workstation. It lets you use more than one logical input device of the same class on a single workstation. For example, you can use a display menu as one choice-class logical input device and a keyboard as another choice-class logical input device.

The device number determines what mechanism triggers the logical input device. (See Section 9.2.5 for information about triggering a logical input device.) DEC GKS defines at least four device numbers for each input class. For example, a choice 1 device number requires the user to press mouse button 1 to trigger the logical input device. (Without a mouse, the user must press Return.) A choice

2 device number requires the user to press the arrow keys or the keys on the numeric keypad.

The device number also determines the format in which DEC GKS returns data. For example, a string 1 device number returns a Digital multinational text string, while a string 3 device number returns an ASCII value.

9.2.2 Controlling the Appearance of the Logical Input Device

The **prompt and echo type** controls what the logical input device looks like on the screen. Each input class has its own set of prompt and echo types. For example, locator-class prompt and echo type 2 marks the current location with cross hairs, while locator-class prompt and echo type 3 marks it with a tracking cross. But valuator-class prompt and echo type 2 displays the current value with a dial or a pointer, while valuator-class prompt and echo type 3 displays a digital representation of the value. (See Section 9.3 for detailed information about prompts and echo types.)

DEC GKS displays the prompt and echo type in the echo area. The echo area cannot be larger than the workstation, but can be smaller than the workstation. The prompt and echo type cursor is active only within the input echo area.

The echo flag controls the visibility of an active logical input device.

9.2.3 Activating and Deactivating a Logical Input Device

You must activate a logical input device before you use it. To activate the logical input device, you must place it in one of three operating modes:

- Request
- Sample
- Event

Request mode is the default operating mode. DEC GKS places the logical input device in request mode when a workstation opens. To activate a logical input device in request mode, call a REQUEST function. DEC GKS activates the logical input device and displays the input prompt (if echoing is enabled) when you call a REQUEST function. For example, to activate a locator-class logical input device in request mode, you would call REQUEST LOCATOR and supply the appropriate values to the arguments. DEC GKS deactivates the logical input device when the REQUEST function completes.

To place a logical input device in request mode, sample mode, or event mode, you must call a SET MODE function and supply the values for the following arguments:

- Workstation identifier
- Device number
- Operating mode (request, sample, or event)
- Echo flag (GECHO or GNECHO)

For example, to place a locator-class logical input device in sample mode, you must call SET LOCATOR MODE and specify SAMPLE as the operating mode.

When DEC GKS places a logical input device in sample mode, it activates the logical input device and displays the input prompt (if echoing is enabled). To have the logical input device return a value to the application, you must call a SAMPLE function. For example, to have a locator-class logical input device in

sample mode return a value, you would call SAMPLE LOCATOR and specify the workstation identifier and the device number.

When DEC GKS places a logical input device in event mode, it activates the logical input device and displays the input prompt (if echoing is enabled). To have the logical input device return a value to the application, you must call the AWAIT EVENT and the GET functions. For example, to have a locator-class logical input device in event mode return a value, you would call AWAIT EVENT. Check the event input class to make sure it is locator, then call GET LOCATOR.

To deactivate a logical input device in sample or event mode, you must place the device back into request mode by calling a SET MODE function and specifying REQUEST as the value for the *MODE* argument. For example, to deactivate a locator-class logical input device in sample mode, you would call SET LOCATOR MODE and specify REQUEST as the operating mode.

9.2.4 Initializing a Logical Input Device

Each workstation has its own default values that a logical input device can use. However, you also can set your own values for the logical input device. To set your own values, you must initialize the logical input device using the INITIALIZE function. The logical input device must be in request mode to be initialized. For example, to initialize a locator-class logical input device, you would put it in request mode by calling SET LOCATOR MODE. Then you would call INITIALIZE LOCATOR and supply the values you want. (See Section 9.4 for detailed information about initializing a logical input device.)

If you do not initialize the logical input device, it uses the default values.

9.2.5 Obtaining Measures from a Logical Input Device

A logical input device returns a value to the application. The value it returns is called the **measure** of the device. Two operating classes, request and event, require the user to perform an action on a physical input device to return the measure. The action is called the **input trigger**. When the user performs the action, the user **triggers** the logical input device, which then returns its measure. The input class and device number determine what kind of action the user must perform to trigger the logical input device. For example, when using a keyboard, the user triggers the logical input device by pressing a key on the keyboard. When using a mouse, the user triggers the logical input device by clicking a mouse button.

Sample mode does not require the user to trigger a logical input device. For example, SAMPLE LOCATOR gets the current value of a locator-class logical input device without any input from the user.

9.2.6 The Input Class

The input class determines the type of input the logical input device returns to the application. You determine the input class when you activate the logical input device. DEC GKS uses six input classes:

- Locator
- Stroke
- Valuator
- Choice

- String
- Pick

A locator-class logical input device first displays a prompt on the workstation surface. The user can then move the prompt and, if the application is using an appropriate input mode, trigger the input device. The locator input class returns two real numbers that represent world coordinate (WC) values. DEC GKS transforms the input point from a device coordinate point to a normalized device coordinate (NDC) point. Then it transforms the NDC point to a corresponding WC point.

A stroke-class logical input device also displays a prompt on the workstation surface. The user can then move the prompt, which causes device coordinate points to be input until the user presses Return. The stroke input class returns a sequence of real numbers that are the corresponding WC values of the stroke. DEC GKS transforms the input points from device coordinate points to NDC points. Then it transforms the NDC points to corresponding WC points.

For more information about the DEC GKS coordinate systems, see Chapter 7, Transformation Functions.

A valuator-class logical input device displays a picture on the workstation surface that represents a series of real numbers. You specify the lowest and highest values in the application. For several workstations, the picture may look like a slide bar with a pointer to a current value. The user moves the cursor up and down the scale to the desired position. The valuator input class returns the real number representing the position of the pointer on the scale.

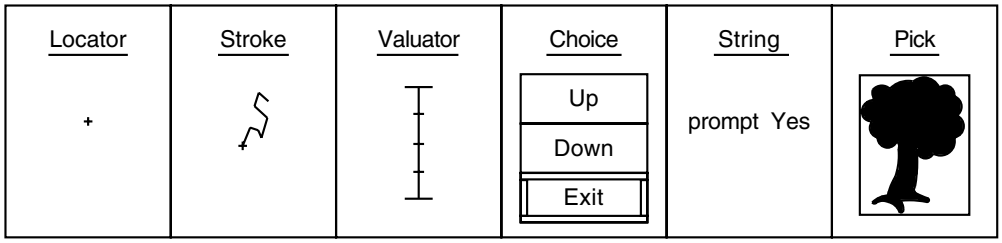
A choice-class logical input device creates a picture on the workstation surface that lists a series of choices. The choices are represented internally by integer values. You can label the choices with text in your application. For several workstations, the choices can look like a menu, with the currently selected choice highlighted. The user moves the choice input prompt through the choices, highlights a choice, and then triggers the device. When the user triggers the choice logical input device, the choice input class returns the integer representing the choice the user selected.

A string-class logical input device displays a prompt on the workstation surface where the user can enter a character string. In the application, you can provide an initial string for the prompt. 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 Return triggers the string-class logical input device. The string input class returns a character string.

A pick-class logical input device positions a prompt on the workstation surface. The user moves the prompt among the segments on the workstation surface. If the application is using an appropriate input mode, the user can trigger the pick device. The pick input class returns integers that represent the name of the picked segment and the **pick identifier** associated with parts of a segment. (For detailed information about pick identifiers, see Chapter 8, Segment Functions.)

Figure 9–1 shows possible visual interfaces for the logical input classes.

Figure 9–1 Visual Interfaces for Logical Input Classes



Significant differences may exist in how workstations implement input classes. For example, using a stroke-class logical input device on a DECwindows workstation, you can specify X and Y device coordinate change vectors to tell the input device when to add another device coordinate point to the stroke. When the user moves the cursor to 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 cursor.

In contrast, on the VT340™ terminal, you must move the arrow keys and signal each time you have reached a point you want to be a part of the stroke. For information on input classes for specific devices, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

9.3 Prompt and Echo Types

A single workstation can prompt the user and echo the input in different ways when using the same logical input 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 MB1. One logical input device can have a number of prompt and echo types. The prompt and echo types provide different visual interfaces for the logical input device. The GKS standard defines some prompt and echo types, while others are implementation dependent.

For example, DEC GKS supports the following prompt types for its locator-class logical input devices:

- A tracking plus sign (+)
- A cross hair
- A tracking cross (X)
- A line from the initial locator position to the current locator position (rubber-band line)
- A rectangle whose diagonal connects the initial and current positions (rubber-band box)
- A numeric representation of the current locator position

The first prompt is implementation dependent. The last five are defined by the GKS standard.

The input devices use DEC GKS primitives such as lines, markers, and fill areas to construct input prompts. However, the input devices may also use additional information that determines the physical appearance of the prompt and input echoed on the surface. For example, an input device may use a polyline output attribute that affects the physical appearance of cross hairs displayed on the surface. The information depends on the needs of the different prompt and echo types on different physical devices. It is provided to the input device through input data records.

9.3.1 DEC GKS Prompt and Echo Types

The following sections describe the prompt and echo types supported by DEC GKS for each class of logical input devices.

Not all prompt and echo types are available for every logical input device with every workstation type. To see which ones are available for a particular workstation type, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

9.3.1.1 Choice-Class Prompt and Echo Types

DEC GKS supports the following choice-class prompt and echo types:

Prompt and Echo Type	Description
–1	Highlights the current choice using a hollow rectangle.
1	Displays the list of choice strings within the echo area.
2	Displays the list of choice strings within the echo area.
3	Displays the list of choice strings within the echo area.

9.3.1.2 Locator-Class Prompt and Echo Types

DEC GKS supports the following locator-class prompt and echo types:

Prompt and Echo Type	Description
–13	Marks the current location using a segment. The segment is drawn relative to the current location. The current location is also marked by a tracking plus sign.
–12	Marks the current location using a rubber-band ellipse centered at the initial point and the current location at the corner of the bounding rectangle.
–11	Marks the current location with the world coordinate translation of the device coordinate position.
–10	Marks the current location using a circle centered at the midpoint of the initial location and the current location.
–9	Marks the current location using a circle centered at the initial position, with the current location on the circumference.
–8	Marks the current location using an open-type arc defined by the current location and two points supplied in the data record.
–7	Marks the current location using a pie-type arc defined by the current location and two points supplied in the data record.

Prompt and Echo Type	Description
–6	Marks the current location using a chord-type arc defined by the current location and two points supplied in the data record.
–5	Marks the current location using a horizontal line drawn from the initial position to the current location.
–4	Marks the current location using a vertical line drawn from the initial position to the current location.
–3	Marks the current location using two lines connected to two fixed points supplied by the data record.
–2	Marks the current location using a rectangle that is centered at the initial points and has a corner at the current location.
–1	Marks the current location with a rectangular box.
1	Marks the current location with a tracking plus sign.
2	Marks the current location by using a vertical and a horizontal line as cross hairs.
3	Marks the current location using a tracking cross.
4	Marks the current location using a line connecting the current location to the initial location (rubber band line).
5	Marks the current location using a rectangle whose diagonal is the line between the current location and the initial location (rubber band box).
6	Marks the current location by displaying a digital representation of the location.

9.3.1.3 Pick-Class Prompt and Echo Types

DEC GKS supports the following pick-class prompt and echo types:

Prompt and Echo Type	Description
1	Highlights the extent rectangle of the picked output primitive.
2	Highlights the extent rectangle of all the output primitives that share the pick identifier of the picked primitive.
3	Highlights the extent rectangle of the picked segment.

9.3.1.4 String-Class Prompt and Echo Type

DEC GKS supports the following string-class prompt and echo types:

Prompt and Echo Type	Description
1	Displays the current string value in the echo area.

9.3.1.5 Stroke-Class Prompt and Echo Types

DEC GKS supports the following stroke-class prompt and echo types:

Prompt and Echo Type	Description
1	Displays a line joining successive points of the current stroke.

Prompt and Echo Type	Description
3	Displays a polymarker at each successive point of the current stroke.
4	Displays a line joining successive points of the current stroke.

9.3.1.6 Valuator-Class Prompt and Echo Types

DEC GKS supports the following valuator-class prompt and echo types:

Prompt and Echo Type	Description
–4	Displays the range as floating values (for use only with the hardware dials).
–3	Displays the range of values in a circular dial (for use only with the VWS workstations).
–2	Displays the range of values on a horizontal sliding scale.
–1	Displays the range of values on a vertical sliding scale.
1	Displays a graphical representation of the current value (such as a dial or a pointer).
2	Displays a graphical representation of the current value (such as a dial or a pointer).
3	Displays a digital representation of the current value.

9.3.2 Input Data Records

If you call one of the INITIALIZE input functions, you must use an **input data record** to pass information about a specific prompt and echo type on a given logical input device. The input data record contains information about the input prompt interface. For example, the input data record for a locator-class logical input device may specify output attributes that affect the thickness or color of cross hairs on the workstation surface. You can use the default input data record or an application-specified input data record. The application-specified input data record is an argument to the INITIALIZE input functions. DEC GKS also uses an input data record to return information to the application with the INQUIRE . . . DEVICE STATE and INQUIRE DEFAULT . . . DEVICE DATA functions. (See the GKS International Standard (ISO 8805(E) 1988) for a detailed description of input data records.)

The GKS standard describes input data records as having required components and optional components. If a component is required, all input devices use that component of the data record. If a component is optional, the input device must be able to accept that component, but may or may not use it when generating the input prompt and echo. For example, suppose a polyline color is an optional part of the data record. The GKS implementation cannot generate an error if it encounters the component and does not have to change the color of the prompt on the workstation surface.

Before you use the INITIALIZE function in the FORTRAN binding to pass an application-defined input data record, you first must pack the input data record using the PACK DATA RECORD function (GPREC). The input data record passed in the INITIALIZE function is the data record that GPREC returns in the *DATREC* argument. The size of the input data record passed in the INITIALIZE function is the size that GPREC returns in the *LDR* argument.

After you call GPREC, use the input data record size (*LDR*) and the input data record argument (*DATREC*) in the INITIALIZE function to affect the prompt and echo of the logical input device. Be sure to check the error indicator argument (*ERRIND*) returned by GPREC before calling the INITIALIZE function.

The tables in the following sections present the input data record information in the form GPREC requires. GPREC uses seven of its ten arguments to pass data record information that needs to be packed:

GPREC Argument	Description
IL	Number of integer entries
RL	Number of real entries
SL	Number of string entries
IA	Array of integers containing the integer entries
RA	Array of reals containing the real entries
LSTR	Array of integers containing the lengths of the string entries
STR	Array of character*80 containing the strings

The following sections list all the input data records for the FORTRAN binding. The tables within these sections include the name of the argument in GPREC, an argument value or a description of the input data record component, and whether the workstation uses (U) or ignores (I) the input data record component. NA means the information is not applicable.

The order of arguments in your arrays *must* match the order listed in the tables. The numbers of integers, reals, and strings in your arrays *must* match those in the tables. If an array does not need to store information, you *must* pass a dummy array. For example, if the number of integers is 0, pass a dummy integer array for the IA argument.

9.3.2.1 Choice Class

This section lists the input data record information for choice-class prompt and echo types.

Choice-Class Prompt and Echo Types –1, 1, and 3

The following table describes how to pack the input data record information using function GPREC for prompt and echo types –1, 1, and 3:

GPREC Argument	Value or Component Description	Used or Ignored
IL	0	NA
RL	0	NA
SL	Number of choice alternatives	U
LSTR	Lengths of the choice strings	U
STR	Choice strings	U

Choice-Class Prompt and Echo Type 2

The following table describes how to pack the input data record information using function GPREC for prompt and echo type 2:

GPREC Argument	Value or Component Description	Used or Ignored
IL	Number of choice alternatives	U
RL	0	NA
SL	0	NA
IA	Array of prompts turned off (GPROFF) or on (GPRON)	U

9.3.2.2 Locator Class

This section lists the input data record information for locator-class prompt and echo types.

Locator-Class Prompt and Echo Type –1

The following table describes how to pack the input data record information using function GPREC for prompt and echo type –1:

GPREC Argument	Value or Component Description	Used or Ignored
IL	0	NA
RL	2	NA
SL	0	NA
RA	X dimension of the box in WC points	U
	Y dimension of the box in WC points	U

Locator-Class Prompt and Echo Types –11, 1, 2, 3, and 6

These prompt and echo types need no input data record information. Pass a dummy input data record to the INITIALIZE LOCATOR function. The size of the input data record, the *LDR* argument in the INITIALIZE LOCATOR function, is 0.

Locator-Class Prompt and Echo Types –12, –10, –9, –5, –4, and 4

These prompt and echo types require you to use one of two input data records. Which input data record you use depends on the value of the attribute control flag. If the attribute control flag is GCURNT, the prompt and echo displays the current set of output attributes. If the attribute control flag is GSPEC, the prompt and echo displays the output attributes specified in the input data record.

To specify the current output attribute settings, pack the input data record information using GPREC as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	1	NA
RL	0	NA
SL	0	NA
IA	Attribute control flag (GCURNT)	U

To specify the new output attribute settings, pack the input data record information using GPREC as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	7	NA
RL	1	NA
SL	0	NA
IA	Attribute control flag (GSPEC)	U
	Line type ASF (GBUNDL or GINDIV)	I
	Line width scale factor ASF (GBUNDL or GINDIV)	I
	Polyline color index ASF (GBUNDL or GINDIV)	I
	Line type index	U
	Polyline color index	I
RA	Line width scale factor	U

Locator-Class Prompt and Echo Types –2 and 5

Although four input data records are theoretically possible for these prompt and echo types, DEC GKS supports only two. The input data records depend on the values of the polyline-fill-area control flag and on the attribute control flag. If the polyline-fill-area control flag is GPLINE, DEC GKS draws as a polyline the prompt and echo rectangle whose diagonal connects the current and the initial points. If the polyline-fill-area control flag is GPLINE, DEC GKS draws the prompt and echo rectangle as a polyline whose diagonal connects the current and initial points. DEC GKS currently supports only the polyline rectangle.

If the attribute control flag is GCURNT, the prompt and echo displays the current set of output attributes. If the attribute control flag is GSPEC, the prompt and echo displays the output attributes specified in the input data record.

To specify the current output attribute values and a polyline rectangle, pack the input data record as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	2	NA
RL	0	NA
SL	0	NA
IA	Polyline-fill-area control flag (GPLINE)	I
	Attribute control flag (GCURNT)	U

To specify new output attributes values and a polyline rectangle, pack the input data record as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	8	NA
RL	1	NA
SL	0	NA

GPREC Argument	Value or Component Description	Used or Ignored
IA	Polyline-fill-area control flag (GPLINE)	I
	Attribute control flag (GSPEC)	U
	Line type ASF (GBUNDL or GINDIV)	I
	Line width scale factor ASF (GBUNDL or GINDIV)	I
	Polyline color index ASF (GBUNDL or GINDIV)	I
	Polyline index	I
	Line type index	U
	Polyline color index	I
RA	Line width scale factor	U

Locator-Class Prompt and Echo Types –8, –7, –6, and –3

These prompt and echo types require you to use one of two input data records. Which input data record you use depends on the value of the attribute control flag. If the attribute control flag is GCURNT, the prompt and echo displays the current set of output attributes. If the attribute control flag is GSPEC, the prompt and echo displays the output attributes specified in the input data record.

To specify the current output attribute settings, pack the input data record information using GPREC as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	1	NA
RL	4	NA
SL	0	NA
IA	Attribute control flag (GCURNT)	U
RA	X component of the first WC point	U
	Y component of the first WC point	U
	X component of the second WC point	U
	Y component of the second WC point	U

To specify the new output attribute settings, pack the input data record information using GPREC as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	7	NA
RL	5	NA
SL	0	NA
IA	Attribute control flag (GSPEC)	U
	Line type ASF (GBUNDL or GINDIV)	I
	Line width scale factor ASF (GBUNDL or GINDIV)	I

GPREC Argument	Value or Component Description	Used or Ignored
	Polyline color ASF (GBUNDL or GINDIV)	I
	Polyline index	I
	Line type index	U
	Polyline color index	I
RA	Line width scale factor	U
	X component of the first WC point	U
	Y component of the first WC point	U
	X component of the second WC point	U
	Y component of the second WC point	U

Locator-Class Prompt and Echo Type –13

The following table describes how to pack the input data record information using function GPREC for prompt and echo type –13:

GPREC Argument	Value or Component Description	Used or Ignored
IL	1	NA
RL	0	NA
SL	0	NA
IA	Segment identifier of the segment used for the cursor segment	U

9.3.2.3 Pick Class

This section lists the input data record information for pick-class prompt and echo types.

Pick-Class Prompt and Echo Types 1, 2, and 3

The following table describes how to pack the input data record information using function GPREC for prompt and echo types 1, 2, and 3:

GPREC Argument	Value or Component Description	Used or Ignored
IL	0	NA
RL	1	NA
SL	0	NA
RA	Size of the pick aperture (prompt) in device coordinates	U

9.3.2.4 String Class

This section lists the input data record information for string-class prompt and echo types.

String-Class Prompt and Echo Type 1

This prompt and echo type needs no input data record information. Pass a dummy input data record in the INITIALIZE STRING function. The size of the input data record, the *LDR* argument in the INITIALIZE STRING function, is 0.

9.3.2.5 Stroke Class

This section lists the input data record information for stroke-class prompt and echo types.

Stroke-Class Prompt and Echo Type 1

The following table describes how to pack the input data record information using function GPREC for prompt and echo type 1:

GPREC Argument	Value or Component Description	Used or Ignored
IL	0	NA
RL	3	NA
SL	0	NA
RA	X component of the WC change vector	U
	Y component of the WC change vector	U
	Time interval in seconds	I

Stroke-Class Prompt and Echo Type 3

This prompt and echo type requires you to use one of two input data records. Which input data record you use depends on the value of the attribute control flag. If the attribute control flag is GCURNT, the prompt and echo displays the current set of output attributes. If the attribute control flag is GSPEC, the prompt and echo displays the output attributes specified in the input data record.

To specify the current output attribute settings, pack the input data record information using GPREC as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	1	NA
RL	3	NA
SL	0	NA
IA	Attribute control flag (GCURNT)	U
RA	X component of the WC change vector	U
	Y component of the WC change vector	U
	Time interval in seconds	I

To specify the new output attribute settings, pack the input data record information using GPREC as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	7	NA
RL	4	NA
SL	0	NA
IA	Attribute control flag (GSPEC)	U
	Polymarker type ASF (GBUNDL or GINDIV)	I

GPREC Argument	Value or Component Description	Used or Ignored
	Polymarker size factor ASF (GBUNDL or GINDIV)	I
	Polymarker color ASF (GBUNDL or GINDIV)	I
	Polymarker index	I
	Polymarker type index	U
	Polymarker color index	I
RA	X component of the WC change vector	U
	Y component of the WC change vector	U
	Time interval in seconds	I
	Polymarker scale factor	U

Stroke-Class Prompt and Echo Type 4

This prompt and echo type requires you to use one of two input data records. Which input data record you use depends on the value of the attribute control flag. If the attribute control flag is GCURNT, the prompt and echo displays the current set of output attributes. If the attribute control flag is GSPEC, the prompt and echo displays the output attributes specified in the input data record.

To specify the current output attribute settings, pack the input data record information using GPREC as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	1	NA
RL	3	NA
SL	0	NA
IA	Attribute control flag (GCURNT)	U
RA	X component of the WC change vector	U
	Y component of the WC change vector	U
	Time interval in seconds	I

To specify the new output attribute settings, pack the input data record information using GPREC as indicated in the following table:

GPREC Argument	Value or Component Description	Used or Ignored
IL	7	NA
RL	4	NA
SL	0	NA
IA	Attribute control flag (GSPEC)	U
	Line type ASF (GBUNDL or GINDIV)	I
	Line width scale factor ASF (GBUNDL or GINDIV)	I
	Polyline color index ASF (GBUNDL or GINDIV)	I

GPREC Argument	Value or Component Description	Used or Ignored
	Polyline index	I
	Line type index	U
	Polyline color index	I
RA	X component of the WC change vector	U
	Y component of the WC change vector	U
	Time interval in seconds	I
	Line width scale factor	U

9.3.2.6 Valuator Class

DEC GKS currently supports six valuator-class prompt and echo types.

Valuator-Class Prompt and Echo Types –3, –2, –1, 1, 2, and 3

These prompt and echo types need no input data record information. Pass a dummy input data record in the INITIALIZE VALUATOR function. The size of the input data record, the *LDR* argument in the INITIALIZE VALUATOR function, is 0.

9.4 Initializing Input

INITIALIZE functions let you specify attributes of the logical input devices. To initialize a logical input device, you must place the device in request mode. Request mode is the DEC GKS default input mode.

After you put the logical input device into request mode, you can either use its default attributes or specify your own. To use the default attributes, activate a logical input device without first calling one of the INITIALIZE functions. To specify your own attributes, call one of the INITIALIZE functions before you activate the logical input device.

INITIALIZE functions include:

- INITIALIZE CHOICE (3)
- INITIALIZE LOCATOR (3)
- INITIALIZE PICK (3)
- INITIALIZE STRING (3)
- INITIALIZE STROKE (3)
- INITIALIZE VALUATOR (3)

9.5 Input Operating Modes

DEC GKS supports three input operating modes: request, sample, and event. You can use any of the six types of logical input devices in any of the three input operating modes.

Some applications must work synchronously with the input process. That is, the application must pause to wait for the user to complete the input action. You can use the request mode to have an application work synchronously.

Some applications must work asynchronously with the input process. That is, the application must run while the user enters input. You can use sample mode or event mode to have an application work asynchronously. In sample mode, the application takes the current measure of an input device without the user having to trigger it. In event mode, the device handler places triggered input values in a time-ordered queue that the application accesses when it needs to process input.

To change the input operating mode for a given device, you call one of the SET MODE functions. Besides changing operating modes, these functions also enable and disable echoing of the input prompt and the input values. Disabling echoing is useful when the DEC GKS echo types are inadequate and you must echo the input in an application-specific manner.

By default, all input prompts are active at once. For example, if you press the arrow keys, you change all input prompts on the workstation surface whose devices use the arrow keys. Device handlers can provide ways for the user to deactivate all but one input prompt, 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. ReGIS™ and some Tektronix® workstations let you cycle through logical input devices. For more information, see the section on cycling logical input devices in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

Note

You cannot cycle past a device whose echoing is disabled. (Normally, DEC GKS notifies you of the logical input device's turn in the cycle by displaying the logical input device's prompt.) Using the corresponding physical device will always change the measure of a nonechoing device. For example, if you use pick-class logical input device 1 on the VT340 terminal while disabling its echoing, pressing the arrow keys always changes the measure of this logical input device no matter how you cycle through the remaining prompting devices.

The following sections describe each of the input operating modes.

9.5.1 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 cancel input. You can use a logical input device in request mode without calling the SET MODE functions if you have not previously set the logical input device to another mode. Request mode is the DEC GKS default input operating mode.

To initialize a logical input device, you must make sure the logical input device's input prompt does not currently appear on the workstation surface. The logical input device must be in request mode to be initialized.

Although you can place any or all the supported logical input devices in request mode at any one time, you can only request input from one logical input device at a time. To request input, you must specify a logical input device number and a workstation identifier and call one of the REQUEST functions. REQUEST functions include the following:

- REQUEST CHOICE
- REQUEST LOCATOR

- REQUEST PICK
- REQUEST STRING
- REQUEST STROKE
- REQUEST VALUATOR

After the application requests input by calling one of the REQUEST functions, DEC GKS displays the input prompt (if echoing is enabled).

In request mode, the user can trigger or break a request for input in several ways. If the user triggers the logical input device, DEC GKS writes the value GOK to the request function *STAT* argument. (See Section 9.2.5 for information about triggering a logical input device.) If the user performs a break requesting input, DEC GKS writes the value GNONE to the request function *STAT* argument. (Different workstations may require different actions for the user to perform a break.)

Choice-class and pick-class logical input devices allow the user another option besides returning data or breaking input. They let the user end the input process without choosing or picking. If the user triggers the logical input device without moving the input prompt, DEC GKS returns one of the appropriate values, GNCHOI or GNPICK, to the *STAT* argument. (DEC GKS also returns GNPICK if the user is not currently positioning the aperture on a segment.)

9.5.2 Sample Mode

In sample mode, the application and the input process operate asynchronously. The user changes the input measure of a given logical input device by changing the position of the input prompt, but cannot trigger the logical input device. The application determines when to sample (take) the current measure of the logical input device. The user specifies input values, but the application controls when it actually accepts the values. The application ends the input session when conditions within the program are met.

To place a logical input device in sample mode, you must specify sample mode to one of the SET MODE functions. As soon as you do, DEC GKS displays the input prompt (if echoing is enabled). At this point, the user can enter input, but cannot trigger the logical input device or cancel input.

To sample input, you must specify a logical input device number and a workstation identifier and call one of the SAMPLE functions. SAMPLE functions include:

- SAMPLE CHOICE
- SAMPLE LOCATOR
- SAMPLE PICK
- SAMPLE STRING
- SAMPLE STROKE
- SAMPLE VALUATOR

After you place the device in sample mode, you cannot reinitialize the device (by calling one of the INITIALIZE functions). If you want to reinitialize the device, you must remove the input prompt from the workstation surface. To remove it, place the device in request mode, reinitialize the device, and then place the device back into sample mode.

You can place any or all the supported logical input devices in sample mode at one time. However, you can sample from only one device at a time. The program can call a SAMPLE function from any point in the application. 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 to request mode.

When sampling choice and pick logical input devices, you can obtain an additional input status. The additional input status can have one of two values: GNCHOI or GNPICK. You can obtain the value GNCHOI if the user did not alter the device's measure since it was activated. You can obtain the value GNPICK if the user did not move the aperture or is not currently positioning the aperture on a segment. Under the specified conditions, DEC GKS writes one of these values to the *STAT* argument.

9.5.3 Event Mode

EVENT functions remove, read, and flush input reports from the event queue. In event mode, the application and the input process operate asynchronously. Event mode differs from sample mode because 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 the user entered.

To place a logical input device in event mode, you must specify event mode to one of the SET functions. As soon as you do, DEC GKS displays the input prompt (if echoing is enabled). At this point, the user can generate events that the device handler places in the event input queue.

EVENT functions include:

- AWAIT EVENT
- FLUSH DEVICE EVENTS
- GET CHOICE
- GET LOCATOR
- GET PICK
- GET STRING
- GET STROKE
- GET VALUATOR

After you place the device in event mode, you cannot reinitialize the device (by calling one of the INITIALIZE functions) until you remove the input prompt from the workstation surface. To remove it, place the device in request mode, reinitialize the device, and then place the device back into event mode.

You can process reports the user generates. To remove a report from the event input queue, call the AWAIT EVENT function. AWAIT EVENT checks the event queue for a length of time up to the amount specified in the *TOUT* argument. If the event queue contains at least one report, AWAIT EVENT removes the oldest report, places it in the *current event report* entry in the GKS state list, and lets the application resume. If the queue remains empty for the entire timeout period, AWAIT EVENT writes GNCLAS to its *ICL* argument and lets the application resume.

Each input report contains the following information that corresponds to the generated event:

- The workstation identifier
- The input class of the device
- The device number
- The input value or values

To process the information in the current event report, you must check the value written to the *ICL* argument of AWAIT EVENT. Once you determine the class of the device that generated the event, you call one of the GET functions.

The GET functions obtain information from the current event report. Therefore, repeated calls to one of the GET functions will write the same values to the output arguments. The current event report does not change unless you call AWAIT EVENT to fetch another report from the queue. After you fetch another report, a subsequent call to one of the GET functions obtains new input values.

If you decide you have enough information from a particular logical input device, you can stop generating events by placing the logical input device back in request mode. Then you can flush all the events the logical input device generated that remain in the event input queue by calling FLUSH DEVICE EVENTS.

Example 9–1 shows a sample program using a locator-class logical input device in event mode.

9.5.3.1 Event Input Queue Overflow

Because the user can generate events as soon as you call a GET function, the user may fill the event input queue before the application can remove any of the event reports. The input event queue could overflow.

If you try to call either AWAIT EVENT or FLUSH DEVICE EVENTS, DEC GKS logs an initial event input queue overflow error (ERROR_147—Input queue has overflowed). If you continue calling either AWAIT EVENT or FLUSH DEVICE EVENTS, the functions still perform their task. However, the logical input devices cannot accept additional input until you clear the input queue. You can generate ERROR_147 many times while trying to clear the queue. DEC GKS, however, logs the error only once, the first time it occurs.

To test for input queue overflow, you can call INQUIRE INPUT QUEUE OVERFLOW immediately after calling AWAIT EVENT. If the *ERRIND* argument to INQUIRE INPUT QUEUE OVERFLOW equals 0, the following is true:

- The event input queue has overflowed.
- Information about the overflow is available.
- INQUIRE INPUT QUEUE OVERFLOW writes to its output arguments the workstation identifier, the input class, and the device number of the logical input device that last accepted input.

If *ERRIND* does not equal 0, the information needed to write to the output arguments is not available. In this case, *ERRIND* can equal one of the following values:

- ERROR_7—GKS not in proper state.
- ERROR_148—Input queue has not overflowed since GKS was opened or since the last invocation of INQUIRE INPUT QUEUE OVERFLOW.

- ERROR_149—Input queue has overflowed, but the associated workstation has been closed.

If the event input queue overflows, you can call FLUSH DEVICE EVENTS to clear the events from the queue. FLUSH DEVICE EVENTS clears the buffer and lets the user enter input again. Because FLUSH DEVICE EVENTS clears individual logical input devices, you must call it for each logical input device the application is using. If you know the input class that caused the overflow, you can call FLUSH DEVICE EVENTS for only that input class. If you do not know which input class caused the overflow, you must call FLUSH DEVICE EVENTS for all the logical input devices and for all the input classes the application was using.

A second way to clear the events from the overflowed queue is to continue calling AWAIT EVENT, removing the reports one by one until a call returns GNCLAS.

Using FLUSH DEVICE EVENTS is the preferred way to clear events from an overflowed queue. If you use AWAIT EVENT, the user may continue generating input faster than AWAIT EVENT can remove events from the queue.

9.6 Overlapping Viewports

This section assumes you know something about the DEC GKS coordinate systems. You may want to review Chapter 7, Transformation Functions, before reading further.

When defining normalization viewports, you may cause them to overlap in NDC space. The overlap can affect the application during input requests. To prevent overlap, the application should use a viewport priority list during input.

To illustrate using a viewport priority list, consider two normalization viewports. The first is the default viewport ([0,1] x [0,1]) of the unity transformation. The second belongs to normalization transformation number 1 and has 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. Then it uses the viewport input priority to determine which normalization transformation to use when translating the points to WC points.

DEC GKS maintains a priority list that it uses to decide which normalization viewport has a higher input priority. 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. For example, viewport 0 is higher than viewport 1; 1 is higher than 2; 2 is higher than 3, and so on.

When using a locator-class input 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 the points in the stroke. A locator or stroke input device cannot return device coordinate points that can fall outside the default normalization viewport ([0,1] x [0,1]). Therefore, you can always use the unity transformation to transform stroke input data.

For more information about transformations and viewport priority, see Chapter 7, Transformation Functions.

9.7 Input Inquiries

When using the DEC GKS input functions, you may need to inquire from the workstation description table or from the workstation state list. If you need default values, you inquire from the description table. If you need the currently set values, you inquire from the state list.

The following sections describe programming techniques for inquiry functions.

9.7.1 Default and Current Input Values

Your application can set all the input values individually before it calls one of the INITIALIZE functions. If you do not want the application to set all the input values, you can have it pass the input values returned by one of two sets of inquiry functions. The first set obtains default input values. The second set obtains current input values.

The following inquiry functions obtain default input values:

- INQUIRE DEFAULT CHOICE DEVICE DATA (3)
- INQUIRE DEFAULT LOCATOR DEVICE DATA (3)
- INQUIRE DEFAULT PICK DEVICE DATA (3)
- INQUIRE DEFAULT STRING DEVICE DATA (3)
- INQUIRE DEFAULT STROKE DEVICE DATA (3)
- INQUIRE DEFAULT VALUATOR DEVICE DATA (3)

The following inquiry functions obtain current input values:

- INQUIRE CHOICE DEVICE STATE (3)
- INQUIRE LOCATOR DEVICE STATE (3)
- INQUIRE PICK DEVICE STATE (3)
- INQUIRE STRING DEVICE STATE (3)
- INQUIRE STROKE DEVICE STATE (3)
- INQUIRE VALUATOR DEVICE STATE (3)

You must use the UNPACK DATA RECORD function after calling an inquiry function if you want to change any of the information passed in the data record. After making the changes, use the PACK DATA RECORD function before calling the INITIALIZE function. Be sure to check the *ERRIND* argument after calling PACK DATA RECORD and UNPACK DATA RECORD.

9.7.2 Device-Independent Programming

You can use the INQUIRE functions when writing device-independent applications. Depending on the type of input you use, you may need to call many INQUIRE functions. For example, your application may 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 the workstation supports.

- The prompt and echo types a given workstation supports.
- 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 9.7.1).

Use the following INQUIRE functions to obtain input information when writing a device-independent application:

INQUIRE CHOICE DEVICE STATE (3)
INQUIRE CURRENT NORMALIZATION TRANSFORMATION NUMBER
INQUIRE DEFAULT CHOICE DEVICE DATA (3)
INQUIRE DEFAULT LOCATOR DEVICE DATA (3)
INQUIRE DEFAULT PICK DEVICE DATA (3)
INQUIRE DEFAULT STRING DEVICE DATA (3)
INQUIRE DEFAULT STROKE DEVICE DATA (3)
INQUIRE DEFAULT VALUATOR DEVICE DATA (3)
INQUIRE DISPLAY SPACE SIZE (3)
INQUIRE INPUT QUEUE OVERFLOW
INQUIRE LEVEL OF GKS
INQUIRE LOCATOR DEVICE STATE (3)
INQUIRE NORMALIZATION TRANSFORMATION (3)
INQUIRE PICK DEVICE STATE (3)
INQUIRE SET OF OPEN WORKSTATIONS
INQUIRE STRING DEVICE STATE (3)
INQUIRE STROKE DEVICE STATE (3)
INQUIRE VALUATOR DEVICE STATE (3)
INQUIRE WORKSTATION CATEGORY
INQUIRE WORKSTATION TRANSFORMATION (3)

For information about device-independent programming, see the *DEC GKS User's Guide*.

9.8 Function Descriptions

This section describes the DEC GKS input functions in detail.

AWAIT EVENT

Operating States

WSOP, WSAC, SGOP

Syntax

GWAIT (TOUT, WKID, ICL, DEVNUM)

Argument	Data Type	Access	Description
TOUT	Real	Read	Time to wait for event
WKID	Integer	Write	Workstation identifier
ICL	Integer (constant)	Write	Input class identifier
DEVNUM	Integer	Write	Logical input device number

Constants

Defined Argument	Constant	Description
ICL	GNCLAS	Input queue is empty
	GLOCAT	Event from a locator device
	GSTROK	Event from a stroke device
	GVALUA	Event from a valuator device
	GCHOIC	Event from a choice device
	GPICK	Event from a pick device
	GSTRIN	Event from a string device
	GVIEW	Event from a viewport device

Description

The AWAIT EVENT function 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 timeout period specified in the timeout argument expires.

The timeout argument is specified in the format *ss.hh*, where *ss* is seconds and *hh* is hundredths of a second. This argument cannot be negative and cannot be larger than 356,400 seconds (99 hours).

If this argument is 0.0, this function allows application execution to continue. It either removes the oldest event or, if there are no events in the queue, returns the value GNCLAS to the input class argument.

When 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 GKS state list
- Writes the event's workstation identifier, input class, and logical device number to its corresponding output arguments

If the timeout period has expired, and if this function finds the queue to be empty, this function writes input class value GNCLAS to its output argument.

If you generate the queue overflow error, this function still performs its task.

See Also

Example 9–1 for a program example using the AWAIT EVENT function

FLUSH DEVICE EVENTS

Operating States

WSOP, WSAC, SGOP

Syntax

GFLUSH (WKID, ICL, DEVNUM)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
ICL	Integer (constant)	Read	Input class identifier
DEVNUM	Integer	Read	Logical input device number

Constants

Defined Argument	Constant	Description
ICL	GNCLAS	Input queue is empty
	GLOCAT	Event from a locator device
	GSTROK	Event from a stroke device
	GVALUA	Event from a valuator device
	GCHOIC	Event from a choice device
	GPICK	Event from a pick device
	GSTRIN	Event from a string device
	GVIEW	Event from a viewport device

Description

The FLUSH DEVICE EVENTS function removes all events generated by the specified logical input device from the input queue. This function performs its task even if it generates the queue overflow error message.

For information about the viewport input class, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*, under the escapes Set Viewport Event and Inquire Viewport Data.

GET CHOICE

Operating States

WSOP, WSAC, SGOP

Syntax

GGTCH (STAT, CHNR)

Argument	Data Type	Access	Description
STAT	Integer (constant)	Write	Status of input device
CHNR	Integer	Write	Choice number

Constants

Defined Argument	Constant	Description
STAT	GOK	Input obtained
	GNCHOI	Device triggered without a choice

Description

The GET CHOICE function obtains information from the *current event report* entry in the GKS state list and writes the choice status and choice value to the output arguments.

If the report contains input generated by anything other than a choice-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

After a successful call to GET CHOICE, the input status parameter contains either the value GOK or GNCHOI.

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET CHOICE MODE

GET LOCATOR

Operating States

WSOP, WSAC, SGOP

Syntax

GGTLC (TNR, LPX, LPY)

Argument	Data Type	Access	Description
TNR	Integer	Write	Normalization transformation number
LPX, LPY	Real	Write	Locator position expressed as a WC point

Description

The GET LOCATOR function obtains information from the *current event report* entry in the GKS state list, and writes the normalization transformation number, and the X and Y WC point values to the output arguments.

If the current event report contains input generated by anything other than a locator-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET LOCATOR MODE
Example 9–1 for a program example using the GET LOCATOR function

GET LOCATOR 3

Operating States

WSOP, WSAC, SGOP

Syntax

GGTLC3 (TNR, IVWIX, PX, PY, PZ)

Argument	Data Type	Access	Description
TNR	Integer	Write	Normalization transformation number
IVWIX	Integer	Write	View index
PX, PY, PZ	Real	Write	Locator position expressed as a WC point

Description

The GET LOCATOR 3 function obtains information from the *current event report* entry in the GKS state list, and writes the normalization transformation number; the view index; and X, Y, and Z WC point values to the output arguments.

If the current event report contains input generated by anything other than a locator-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

The GET LOCATOR 3 function returns the view index of the viewport mapping transformation last used to translate the NPC points to the NDC points. See the *DEC GKS User's Guide* for more information on view indexes.

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET LOCATOR MODE
Example 9–1 for a program example using the GET LOCATOR function

GET PICK

Operating States

WSOP, WSAC, SGOP

Syntax

GGTPK (STAT, SGNA, PKID)

Argument	Data Type	Access	Description
STAT	Integer (constant)	Write	Status of input value
SGNA	Integer	Write	Segment name
PKID	Integer	Write	Pick identifier

Constants

Defined Argument	Constant	Description
STAT	GOK	Input obtained
	GNPICK	Device triggered without picking

Description

The GET PICK function obtains information from the *current event report* entry in the GKS state list and writes the input status, segment name, and pick identifier to the output arguments.

If the current event report contains input generated by anything other than a pick-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET PICK MODE

GET STRING

Operating States

WSOP, WSAC, SGOP

Syntax

GGTST (LOSTR, STR)

Argument	Data Type	Access	Description
LOSTR	Integer	Write	Number of characters returned
STR	Character*(*)	Write	String of characters

Description

The GET STRING function obtains information from the *current event report* entry in the GKS state list and writes the string, the string size, and the number of characters written to the string output argument.

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 this function
- The logical input device's string buffer, whose size you can specify in the call to the INITIALIZE STRING function

When reading a string from the current event report using the GET STRING function, DEC GKS removes characters up to the number that fits 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 GET STRING again, using a larger application buffer, to obtain the entire string contained in the report. (Remember that the string contained in the current report does not change until you call the AWAIT EVENT function to replace the current report.)

If the current event report contains input generated by anything other than a string-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

Note

The initial string appears only in the first generated string event report. Subsequent string reports do not contain the initial string.

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET STRING MODE

GET STRING (FORTRAN–77 Subset)

Operating States

WSOP, WSAC, SGOP

Syntax

GGTST (LOSTR, STR)

Argument	Data Type	Access	Description
LOSTR	Integer	Write	Number of characters returned
STR	Character*80	Write	String of characters

Description

The GET STRING function obtains information from the *current event report* entry in the GKS state list and writes the string, the string size, and the number of characters written to the string output argument.

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 this function
- The logical input device's string buffer, whose size you can specify in the call to the INITIALIZE STRING function

When reading a string from the current event report using the GET STRING function, DEC GKS removes characters up to the number that fits 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 GET STRING again, using a larger application buffer, to obtain the entire string contained in the report. (Remember that the string contained in the current report does not change until you call the AWAIT EVENT function to replace the current report.)

If the current event report contains input generated by anything other than a string-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

Note

The initial string appears only in the first generated string event report. Subsequent string reports do not contain the initial string.

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET STRING MODE

GET STROKE

Operating States

WSOP, WSAC, SGOP

Syntax

GGTSK (N, TNR, NP, PXA, PYA)

Argument	Data Type	Access	Description
N	Integer	Read	Dimension of arrays for stroke points
TNR	Integer	Write	Normalization transformation number
NP	Integer	Write	Number of points returned
PXA(N), PYA(N)	Array of reals	Write	WC points comprising the stroke

Description

The GET STROKE function obtains information from the *current event report* entry in the GKS state list and writes the normalization transformation number, 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, which you allocate before the call. You must specify the size in the *N* argument.
- The logical input device's stroke buffer, whose size you can specify in the call to the INITIALIZE STROKE function.

When reading stroke points from the current event report using the GET STROKE function, DEC GKS removes points up to the number that fits 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 GET STROKE again, using a larger application buffer, to obtain the entire stroke contained in the report. (Remember that the stroke contained in the current report does not change until you call the AWAIT EVENT function to replace the current report.)

If the current event report contains input generated by anything other than a stroke-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

Note

The initial stroke appears only in the first generated stroke event report. Subsequent stroke reports do not contain the initial stroke.

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET STROKE MODE

GET STROKE 3

Operating States

WSOP, WSAC, SGOP

Syntax

GGTSK3 (N, TNR, VIEWI, NP, SX, SY, SZ)

Argument	Data Type	Access	Description
N	Integer	Read	Dimension of arrays of stroke points
TNR	Integer	Write	Normalization transformation number
VIEWI	Integer	Write	View index
NP	Integer	Write	Number of points returned
SX(N), SY(N), SZ(N)	Array of reals	Write	WC points comprising the stroke

Description

The GET STROKE 3 function obtains information from the *current event report* entry in the GKS state list and writes the normalization transformation number, the number of entered points, the stroke point values, the number of accepted stroke point values, and the view index used to convert NPC points to NDC points to the output arguments.

When activating stroke input, the following two buffers exist:

- The application's stroke buffer, which you allocate before the call. You must specify the size in the *N* argument.
- The logical input device's stroke buffer, whose size you can specify in the call to the INITIALIZE STROKE 3 function.

When reading stroke points from the current event report using the GET STROKE 3 function, DEC GKS removes points up to the number that fits 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 GET STROKE 3 again, using a larger application buffer, to obtain the entire stroke contained in the report. (Remember that the stroke contained in the current report does not change until you call the AWAIT EVENT function to replace the current report.)

If the current event report contains input generated by anything other than a stroke-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

Note

The initial stroke appears only in the first generated stroke event report. Subsequent stroke reports do not contain the initial stroke.

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET STROKE MODE

GET VALUATOR

Operating States

WSOP, WSAC, SGOP

Syntax

GGTVL (VAL)

Argument	Data Type	Access	Description
VAL	Real	Write	Measure of valuator device

Description

The GET VALUATOR function obtains the valuator input value from the *current event report* entry in the GKS state list and writes the real value to the output argument.

If the current event report contains input generated by anything other than a valuator-class logical input device, a call to this function generates an error. (See the AWAIT EVENT function in this chapter for more information concerning device class and the *current event report* entry.)

See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET VALUATOR MODE

INITIALIZE CHOICE

Operating States

WSOP, WSAC, SGOP

Syntax

GINCH (WKID, DEVNUM, ISTAT, ICHNR, PET, XMIN, XMAX, YMIN, YMAX, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Choice device number.
ISTAT	Integer (constant)	Read	Initial choice status of input device.
ICHNR	Integer	Read	Initial choice number.
PET	Integer	Read	Prompt and echo type.
XMIN, XMAX, YMIN, YMAX	Real	Read	Echo area in device coordinates.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Constants

Defined Argument	Constant	Description
ISTAT	GOK	The initial segment and pick identifier are chosen.
	GNCHOI	No segment or choice identifier is returned.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE CHOICE function establishes the initial values of a choice-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 choice 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 INITIALIZE CHOICE before you request input from a choice-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET CHOICE MODE
Example 9–3 for a program example using an INITIALIZE . . . function

INITIALIZE CHOICE 3

Operating States

WSOP, WSAC, SGOP

Syntax

GINCH3 (WKID, DEVNUM, ISTAT, ICHNR, PET, EVOL, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Choice device number.
ISTAT	Integer (constant)	Read	Initial choice status of input device.
ICHNR	Integer	Read	Initial highlighted choice.
PET	Integer	Read	Prompt and echo type.
EVOL(6)	Array of reals	Read	Echo volume in device coordinates, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Constants

Defined Argument	Constant	Description
ISTAT	GOK	The initial segment and pick identifier are chosen.
	GNCHOI	No segment or choice identifier is returned.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE CHOICE 3 function establishes the initial values of a choice-class logical input device only if the device's prompt is not currently 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 volume, and the data record. Subsequent requests for choice input use the values you specify.

If you do not call INITIALIZE CHOICE 3 before you request input from a choice-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET CHOICE MODE
Example 9–3 for a program example using an INITIALIZE . . . function

INITIALIZE LOCATOR

Operating States

WSOP, WSAC, SGOP

Syntax

GINLC (WKID, DEVNUM, TNR, IPX, IPY, PET, XMIN, XMAX, YMIN, YMAX, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Locator device number.
TNR	Integer	Read	Normalization transformation number.
IPX, IPY	Real	Read	Initial prompt position expressed as a WC point.
PET	Integer	Read	Prompt and echo type.
XMIN, XMAX, YMIN, YMAX	Real	Read	Echo area in device coordinates.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE LOCATOR function 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 WC position 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, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

If you do not call INITIALIZE LOCATOR before you request input from a locator-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET LOCATOR MODE
SET VIEWPORT INPUT PRIORITY
Example 9–1 for a program example using the INITIALIZE LOCATOR function

INITIALIZE LOCATOR 3

Operating States

WSOP, WSAC, SGOP

Syntax

GINLC3 (WKID, DEVNUM, ITNR, IVWIX, IPX, IPY, IPZ, PET, EVOL, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Locator device number.
ITNR	Integer	Read	Normalization transformation number.
IVWIX	Integer	Read	Initial view index.
IPX, IPY, IPZ	Real	Read	Initial prompt position expressed as a WC point.
PET	Integer	Read	Prompt and echo type.
EVOL(6)	Array of reals	Read	Echo volume in device coordinates, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE LOCATOR 3 function 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 WC position of the initial locator, the normalization transformation used to transform the initial locator point, the initial view index, the prompt and echo type, the echo volume, and the data record. Subsequent requests for locator input use the values you specify.

For more information about the locator position and echo types, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

If you do not call INITIALIZE LOCATOR 3 before you request input from a locator-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET LOCATOR MODE
SET VIEWPORT INPUT PRIORITY
Example 9–1 for a program example using the INITIALIZE LOCATOR function

INITIALIZE PICK

Operating States

WSOP, WSAC, SGOP

Syntax

GINPK (WKID, DEVNUM, ISTAT, ISGNA, IPKID, PET, XMIN, XMAX, YMIN, YMAX, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Pick device number.
ISTAT	Integer (constant)	Read	Initial status of logical input device.
ISGNA	Integer	Read	Initial segment name.
IPKID	Integer	Read	Initial pick identifier at which the prompt is placed.
PET	Integer	Read	Prompt and echo type.
XMIN, XMAX, YMIN, YMAX	Real	Read	Echo area in device coordinates.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Constants

Defined Argument	Constant	Description
ISTAT	GOK	The initial segment and pick identifier are chosen.
	GNPICK	No segment or pick identifier is returned.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE PICK 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 prompt and echo type, the echo area, the data record, and the initial pick identifier. A pick identifier is an integer that represents a portion of a segment, allowing you to pick subsets of a segment instead of picking the entire segment. Subsequent requests for pick input use the values you specify.

If you do not call INITIALIZE PICK before you request input from a pick-class logical input device, DEC GKS uses the default input values. For more information concerning the default input values, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

PACK DATA RECORD
SET PICK MODE
Example 9–2 for a program example using the INITIALIZE PICK function

INITIALIZE PICK 3

Operating States

WSOP, WSAC, SGOP

Syntax

GINPK3 (WKID, DEVNUM, ISTAT, ISGNA, IPKID, PET, EVOL, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Pick device number.
ISTAT	Integer (constant)	Read	Initial status of logical input device.
ISGNA	Integer	Read	Initial segment name.
IPKID	Integer	Read	Initial pick identifier at which the prompt is placed.
PET	Integer	Read	Prompt and echo type.
EVOL(6)	Array of reals	Read	Echo volume in device coordinates, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Constants

Defined Argument	Constant	Description
ISTAT	GOK	The initial segment and pick identifier are chosen.
	GNPICK	No segment or pick identifier is returned.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE PICK 3 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 prompt and echo type, the echo volume, the data record, and the initial pick identifier. A pick identifier is an integer that represents a portion of a segment, allowing you to pick subsets of a segment instead of picking the entire segment. Subsequent requests for pick input use the values you specify.

If you do not call INITIALIZE PICK 3 before you request input from a pick-class logical input device, DEC GKS uses the default input values. For more information concerning the default input values, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

PACK DATA RECORD
SET PICK MODE
Example 9–2 for a program example using the INITIALIZE PICK function

INITIALIZE STRING

Operating States

WSOP, WSAC, SGOP

Syntax

GINST (WKID, DEVNUM, LSTR, ISTR, PET, XMIN, XMAX, YMIN, YMAX, BUFLEN, INIPOS, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	String device number.
LSTR	Integer	Read	Length of the initial string.
ISTR	Character*(*)	Read	Initial string. When input has been requested, the user can delete or edit the initial string; otherwise, the newly input string is appended to the initial string.
PET	Integer	Read	Prompt and echo type.
XMIN, XMAX, YMIN, YMAX	Real	Read	Echo area in device coordinates.
BUFLEN	Integer	Read	Size of input buffer.
INIPOS	Integer	Read	Initial cursor position.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE STRING function 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 string input use the values you specify.

If you do not call INITIALIZE STRING before you request input from the string-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET STRING MODE
Example 9–3 for a program example using the INITIALIZE STRING function

INITIALIZE STRING (FORTRAN–77 Subset)

Operating States

WSOP, WSAC, SGOP

Syntax

GINST (WKID, DEVNUM, LSTR, ISTR, PET, XMIN, XMAX, YMIN, YMAX, BUFLEN, INIPOS, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	String device number.
LSTR	Integer	Read	Length of initial string.
ISTR	Character*80	Read	Initial string. When input has been requested, the user can delete or edit the initial string; otherwise, the newly input string is appended to the initial string.
PET	Integer	Read	Prompt and echo type.
XMIN, XMAX, YMIN, YMAX	Real	Read	Echo area in device coordinates.
BUFLEN	Integer	Read	Size of input buffer.
INIPOS	Integer	Read	Initial cursor position.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE STRING function 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 string input use the values you specify.

If you do not call INITIALIZE STRING before you request input from the string-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET STRING MODE
Example 9–3 for a program example using the INITIALIZE STRING function

INITIALIZE STRING 3

Operating States

WSOP, WSAC, SGOP

Syntax

GINST3 (WKID, DEVNUM, LSTR, ISTR, PET, EVOL, BUFLEN, INIPOS, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	String device number.
LSTR	Integer	Read	Length of the initial string.
ISTR	Character*(*)	Read	Initial string. When input has been requested, the user can delete or edit the initial string. Otherwise the newly input string is appended to the initial string.
PET	Integer	Read	Prompt and echo type.
EVOL(6)	Array of reals	Read	Echo volume in device coordinates, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.
BUFLEN	Integer	Read	Size of input buffer.
INIPOS	Integer	Read	Initial cursor position.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE STRING 3 function 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 volume, and the data record. Subsequent requests for string input use the values you specify.

If you do not call INITIALIZE STRING 3 before you request input from the string-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET STRING MODE
Example 9–3 for a program example using the INITIALIZE STRING function

INITIALIZE STRING 3 (FORTRAN–77 Subset)

Operating States

WSOP, WSAC, SGOP

Syntax

GINST3 (WKID, DEVNUM, LSTR, ISTR, PET, EVOL, BUFLEN, INIPOS, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	String device number.
LSTR	Integer	Read	Length of initial string.
ISTR	Character*80	Read	Initial string. When input has been requested, the user can delete or edit the initial string. Otherwise the newly input string is appended to the initial string.
PET	Integer	Read	Prompt and echo type.
EVOL(6)	Array of reals	Read	Echo volume in device coordinates, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.
BUFLEN	Integer	Read	Size of input buffer.
INIPOS	Integer	Read	Initial cursor position.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE STRING 3 function 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 volume, and the data record. Subsequent requests for string input use the values you specify.

If you do not call INITIALIZE STRING 3 before you request input from the string-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET STRING MODE
Example 9–3 for a program example using the INITIALIZE STRING function

INITIALIZE STROKE

Operating States

WSOP, WSAC, SGOP

Syntax

GINSK (WKID, DEVNUM, ITNR, N, IPX, IPY, PET, XMIN, XMAX, YMIN, YMAX, BUFLEN, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Stroke device number.
ITNR	Integer	Read	Normalization transformation number.
N	Integer	Read	Number of points in the initial stroke.
IPX(*), IPY(*)	Array of reals	Read	WC points in the initial stroke.
PET	Integer	Read	Prompt and echo type.
XMIN, XMAX, YMIN, YMAX	Real	Read	Echo area in device coordinates.
BUFLEN	Integer	Read	Size of input buffer.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE STROKE function 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 WC points in the initial stroke, the normalization transformation number used to translate WC points of the initial stroke to NDC points, the prompt and echo

type, the echo area, and the data record. Subsequent requests for stroke input use the values you specify.

If you do not call INITIALIZE STROKE before you request input from a stroke-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET STROKE MODE
SET VIEWPORT INPUT PRIORITY
Example 9–3 for a program example using an INITIALIZE . . . function

INITIALIZE STROKE 3

Operating States

WSOP, WSAC, SGOP

Syntax

GINSK3 (WKID, DEVNUM, ITNR, IVIEWI, N, ISX, ISY, ISZ, PET, EVOL, BUFLEN, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Stroke device number.
ITNR	Integer	Read	Normalization transformation number.
IVIEWI	Integer	Read	View index.
N	Integer	Read	Number of points in the initial stroke.
ISX(*), ISY(*), ISZ(*)	Array of reals	Read	WC points in the initial stroke.
PET	Integer	Read	Prompt and echo type.
EVOL(6)	Array of reals	Read	Echo volume in device coordinates, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.
BUFLEN	Integer	Read	Size of input buffer.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE STROKE 3 function 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 WC values in the initial stroke, the normalization transformation number used to translate WC points of the initial stroke to NDC points, the initial view index, the prompt and echo type, the echo volume, and the data record. Subsequent requests for stroke input use the values you specify.

If you do not call INITIALIZE STROKE 3 before you request input from a stroke-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET STROKE MODE
SET VIEWPORT INPUT PRIORITY
Example 9–3 for a program example using an INITIALIZE . . . function

INITIALIZE VALUATOR

Operating States

WSOP, WSAC, SGOP

Syntax

GINVL (WKID, DEVNUM, IVAL, PET, XMIN, XMAX, YMIN, YMAX, LOVAL, HIVAL, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Valuator device number.
IVAL	Real	Read	Initial valuator value.
PET	Integer	Read	Prompt and echo type.
XMIN, XMAX, YMIN, YMAX	Real	Read	Echo area in device coordinates.
LOVAL, HIVAL	Real	Read	Minimum and maximum valuator values.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE VALUATOR function 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 valuator input use the values you specify.

If you do not call INITIALIZE VALUATOR before you request input from a valuator-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET VALUATOR MODE
Example 9–4 for a program example using the INITIALIZE VALUATOR function

INITIALIZE VALUATOR 3

Operating States

WSOP, WSAC, SGOP

Syntax

GINVL3 (WKID, DEVNUM, IVAL, PET, EVOL, LOVAL, HIVAL, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier.
DEVNUM	Integer	Read	Valuator device number.
IVAL	Real	Read	Initial valuator value.
PET	Integer	Read	Prompt and echo type.
EVOL(6)	Array of reals	Read	Echo volume in device coordinates, in the order XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX.
LOVAL, HIVAL	Real	Read	Minimum and maximum valuator values.
LDR	Integer	Read	Number of elements in the data record. This argument would typically be set to the value of the *LDR* argument returned by the PACK DATA RECORD function.
DATREC(LDR)	Character*80	Read	Data record constructed by the PACK DATA RECORD function.

Note

See the appendix on input values in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for information on prompt and echo types, and data record format.

Description

The INITIALIZE VALUATOR 3 function 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 volume, and the data record. Subsequent requests for valuator input use the values you specify.

If you do not call INITIALIZE VALUATOR 3 before you request input from a valuator-class logical input device, DEC GKS uses the default input values.

See Also

PACK DATA RECORD
SET VALUATOR MODE
Example 9–4 for a program example using the INITIALIZE VALUATOR function

PACK DATA RECORD

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GPREC (IL, IA, RL, RA, SL, LSTR, STR, MLDR, ERRIND, LDR, DATREC)

Argument	Data Type	Access	Description
IL	Integer	Read	Number of integer entries.
IA(*)	Array of integers	Read	Array containing the integer entries.
RL	Integer	Read	Number of real entries.
RA(*)	Array of reals	Read	Array containing the real entries.
SL	Integer	Read	Number of string entries.
LSTR(*)	Array of integers	Read	Array containing the lengths of the character string entries.
STR(*)	Character*80	Read	Array containing the character string entries. Each element in the array should be of dimension 80. For example, character*80 *STR(5)* is a 5-element array.
MLDR	Integer	Read	Number of elements in *DATREC* array.
ERRIND	Integer	Write	Error indicator.
LDR	Integer	Write	Number of elements used in *DATREC* array.
DATREC(MLDR)	Character*80	Write	Data record.

Description

The PACK DATA RECORD function packs a data record. This function does not generate an error. It reports error conditions through the *ERRIND* argument.

PACK DATA RECORD (FORTRAN–77 Subset)

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GPREC (IL, IA, RL, RA, SL, LSTR, STR, MLDR, ERRIND, LDR, DATREC)

Argument	Data Type	Access	Description
IL	Integer	Read	Number of integer entries.
IA(*)	Array of integers	Read	Array containing the integer entries.
RL	Integer	Read	Number of real entries.
RA(*)	Array of reals	Read	Array containing the real entries.
SL	Integer	Read	Number of string entries.
LSTR(*)	Array of integers	Read	Array containing the lengths of the character string entries.
STR(*)	Character*80	Read	Array containing the character string entries. Each element in the array should be of dimension 80. For example, character*80 *STR(5)* is a 5-element array.
MLDR	Integer	Read	Number of elements in *DATREC* array.
ERRIND	Integer	Write	Error indicator.
LDR	Integer	Write	Number of elements used in *DATREC* array.
DATREC(MLDR)	Character*80	Write	Data record.

Description

The PACK DATA RECORD function packs a data record. This function does not generate an error. It reports error conditions through the *ERRIND* argument.

REQUEST CHOICE

Operating States

WSOP, WSAC, SGOP

Syntax

GRQCH (WKID, DEVNUM, STAT, CHNR)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Choice device number
STAT	Integer (constant)	Write	Status of input device
CHNR	Integer	Write	User's choice number

Constants

Defined Argument	Constant	Description
STAT	GNONE	Input break
	GOK	Input obtained
	GNCHOI	Device triggered without choosing

Description

The REQUEST CHOICE function prompts the user for input according to the specifications passed to the INITIALIZE CHOICE and SET CHOICE MODE functions, and returns the status and measure of the response.

If the user enters input, the function writes 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 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 OK indicates input; and the status NOCHOICE indicates that the user did not make a choice (input was triggered without the cursor being moved).

See Also

INITIALIZE CHOICE
SET CHOICE MODE
Example 9–3 for a program example using a REQUEST . . . function

REQUEST LOCATOR

Operating States

WSOP, WSAC, SGOP

Syntax

GRQLC (WKID, DEVNUM, STAT, TNR, PX, PY)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Locator device number
STAT	Integer (constant)	Write	Status of input device
TNR	Integer	Write	Normalization transformation number
PX, PY	Real	Write	Locator position expressed as a WC point

Constants

Defined Argument	Constant	Description
STAT	GNONE	No input obtained
	GOK	Input obtained

Description

The REQUEST LOCATOR function prompts the user for input according to the specifications passed to the INITIALIZE LOCATOR and SET LOCATOR MODE functions, and returns the status and measure of the response.

If the user enters input, the function writes OK to the status argument and writes the locator information to the output arguments. This information includes the transformation number used to transform the device coordinate to a WC point, and the corresponding WC point.

If the user invokes a break action, the function writes NONE to the status argument and the input values are not valid.

For more information about the locator position and PETs, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

See Also

INITIALIZE LOCATOR
SET LOCATOR MODE
Example 9–3 for a program example using a REQUEST . . . function

REQUEST LOCATOR 3

Operating States

WSOP, WSAC, SGOP

Syntax

GRQLC3 (WKID, DEVNUM, STAT, TNR, VIEWI, PX, PY, PZ)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Locator device number
STAT	Integer (constant)	Write	Status of input device
TNR	Integer	Write	Normalization transformation number
VIEWI	Integer	Write	View index
PX, PY, PZ	Real	Write	Locator position expressed as a WC point

Constants

Defined Argument	Constant	Description
STAT	GNONE	No input obtained
	GOK	Input obtained

Description

The REQUEST LOCATOR 3 function prompts the user for input according to the specifications passed to the INITIALIZE LOCATOR 3 and SET LOCATOR MODE functions, and returns the status and measure of the response.

If the user invokes a break action, the function writes NONE to the status argument, and the input values are not valid.

If the user enters input, the function writes OK to the status argument and writes the locator information to the output arguments. The information returned by the REQUEST LOCATOR 3 function includes the locator position expressed as a in WC point, the normalization transformation number used in the conversion to a WC point, and the view index used in the conversion from an NPC point to an NDC point.

See Also

INITIALIZE LOCATOR 3
SET LOCATOR MODE
Example 9–3 for a program example using a REQUEST . . . function

REQUEST PICK

Operating States

WSOP, WSAC, SGOP

Syntax

GRQPK (WKID, DEVNUM, STAT, SGNA, PKID)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Pick device number
STAT	Integer (constant)	Write	Status of input device
SGNA	Integer	Write	Segment name
PKID	Integer	Write	Pick identifier of primitive

Constants

Defined Argument	Constant	Description
STAT	GNONE	Break during input
	GOK	Input obtained
	GNPICK	Triggered without picking

Description

The REQUEST PICK function prompts the user for input according to the specifications passed to the INITIALIZE PICK and SET PICK MODE functions, and returns the status and measure of the response.

If the user enters the input, the function writes OK to the status argument, and writes the integers representing the name of the chosen segment and the chosen pick identifier (see the SET PICK IDENTIFIER function) to the output arguments.

If the user invokes a break action, the function returns NONE to the status argument, and the input values are not valid. 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, this function writes NOPICK to the status argument.

See Also

INITIALIZE PICK
SET PICK IDENTIFIER
SET PICK MODE
Example 9–3 for a program example using a REQUEST . . . function

REQUEST STRING

Operating States

WSOP, WSAC, SGOP

Syntax

GRQST (WKID, DEVNUM, STAT, LOSTR, STR)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	String device number
STAT	Integer (constant)	Write	Status of input device
LOSTR	Integer	Write	Length of returned string
STR	Character*(*)	Write	Input character string

Constants

Defined Argument	Constant	Description
STAT	GNONE	No input obtained
	GOK	Input obtained

Description

The REQUEST STRING function prompts the user for input according to the specifications passed to the INITIALIZE STRING and SET STRING MODE functions, and returns the status and measure of the response.

When you call this function, the following two buffers exist:

- The application's string buffer, which you allocate before the call. You must specify the size to be at least one byte larger than the maximum possible string size. It is best to specify a string buffer size of 256 characters.
- The logical input device's string buffer, whose size you can specify in the call to the INITIALIZE STRING function.

If the user enters input, the function writes OK to the status argument, the character string to the application's buffer, and the length of the character string to the *LOSTR* 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 NONE to the status argument, and the input arguments are not valid.

See Also

INITIALIZE STRING
SET STRING MODE
Example 9–3 for a program example using the REQUEST STRING function

REQUEST STRING (FORTRAN–77 Subset)

Operating States

WSOP, WSAC, SGOP

Syntax

GRQST (WKID, DEVNUM, STAT, LOSTR, STR)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	String device number
STAT	Integer (constant)	Write	Status of input device
LOSTR	Integer	Write	Length of returned string
STR	Character*80	Write	Input character string

Constants

Defined Argument	Constant	Description
STAT	GNONE	No input obtained
	GOK	Input obtained

Description

The REQUEST STRING function prompts the user for input according to the specifications passed to the INITIALIZE STRING and SET STRING MODE functions, and returns the status and measure of the response.

When you call this function, the following two buffers exist:

- The application's string buffer, which you allocate before the call. You must specify the size to be at least one byte larger than the maximum possible string size. It is best to specify a string buffer size of 256 characters.
- The logical input device's string buffer, whose size you can specify in the call to the INITIALIZE STRING function.

If the user enters input, the function writes OK to the status argument, the character string to the application's buffer, and the length of the character string to the *LOSTR* 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 NONE to the status argument, and the input arguments are not valid.

See Also

INITIALIZE STRING
SET STRING MODE
Example 9–3 for a program example using a REQUEST STRING function

REQUEST STROKE

Operating States

WSOP, WSAC, SGOP

Syntax

GRQSK (WKID, DEVNUM, N, STAT, TNR, NP, PXA, PYA)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Stroke device number
N	Integer	Read	Dimension of arrays for stroke points
STAT	Integer (constant)	Write	Status of input device
TNR	Integer	Write	Normalization transformation number
NP	Integer	Write	Number of points in the stroke
PXA(N), PYA(N)	Array of reals	Write	WC points in the stroke

Constants

Defined Argument	Constant	Description
STAT	GNONE	No input obtained
	GOK	Input obtained

Description

The REQUEST STROKE function prompts the user for input according to the specifications passed to the INITIALIZE STROKE and SET STROKE MODE functions, and returns the status and measure of the response.

If the user enters input, the function writes OK to the status argument, and writes the normalization transformation number used to translate the device coordinate points to WC points, the returned stroke points, the total number of entered points, and the number of returned points as output arguments.

When you call this function, the following two buffers exist:

- The application's stroke buffer, which you allocate before the call. You must specify the buffer size in the *N* argument.
- The logical input device's stroke buffer, whose size you can specify in the call to the INITIALIZE STROKE function.

DEC GKS can return points up to 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 buffer, you lose all additional data. You must make sure that your application's buffer is as large as the device's stroke buffer.

If the user invokes a break action, the function returns NONE to the status argument, and the input values are not valid.

See Also

INITIALIZE STROKE
SET STROKE MODE
Example 9–3 for a program example using a REQUEST . . . function

REQUEST STROKE 3

Operating States

WSOP, WSAC, SGOP

Syntax

GRQSK3 (WKID, DEVNUM, N, STAT, TNR, VIEWI, NP, SX, SY, SZ)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Stroke device number
N	Integer	Read	Size of arrays for stroke points
STAT	Integer (constant)	Write	Status of input device
TNR	Integer	Write	Normalization transformation number
VIEWI	Integer	Write	View index
NP	Integer	Write	Number of points in the stroke
SX(N), SY(N), SZ(N)	Array of reals	Write	WC points in the stroke

Constants

Defined Argument	Constant	Description
STAT	GNONE	No input obtained
	GOK	Input obtained

Description

The REQUEST STROKE 3 function prompts the user for input according to the specifications passed to the INITIALIZE STROKE 3 and SET STROKE MODE functions, and returns the status and measure of the response.

If the user enters input, the function writes OK to the status argument and writes the normalization transformation number used to translate the device coordinates to WC points. It also writes the view index, the returned stroke points, the total number of entered points and the number of returned points as output arguments.

When you call this function, two buffers exist:

- The application's stroke buffer, which you allocate before the call. You must specify the buffer size in the *N* argument.
- The logical input device's stroke buffer, whose size is specified in the call to the INITIALIZE STROKE 3 function.

DEC GKS can return points up to the size of the application's X, Y, and Z coordinate buffers. If the size of the entered stroke is larger than the number of points placed in the application's buffer, you lose all additional data. You must make sure that your application's buffer is as large as the device's stroke buffer.

If the user invokes a break action, the function returns NONE to the status argument and the input values are not valid.

See Also

INITIALIZE STROKE 3
SET STROKE MODE
Example 9–3 for a program example using a REQUEST . . . function

REQUEST VALUATOR

Operating States

WSOP, WSAC, SGOP

Syntax

GRQVL (WKID, DEVNUM, STAT, VAL)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Valuator device number
STAT	Integer (constant)	Write	Status of input device
VAL	Real	Write	Logical input value that is the current measure of the device

Constants

Defined Argument	Constant	Description
STAT	GNONE	No input obtained
	GOK	Input obtained

Description

The REQUEST VALUATOR function prompts the user for input according to the specifications passed to the INITIALIZE VALUATOR and SET VALUATOR MODE functions, and returns the status and measure of the response.

If the user accepts the input, the function writes OK to the status argument, and the selected real number to the valuator data.

If the user invokes a break action, the function returns NONE to the status argument, and the input value is not valid.

See Also

INITIALIZE VALUATOR
SET VALUATOR MODE
Example 9–3 for a program example using a REQUEST . . . function

SAMPLE CHOICE

Operating States

WSOP, WSAC, SGOP

Syntax

GSMCH (WKID, DEVNUM, STAT, CHNR)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Choice device number
STAT	Integer (constant)	Write	Status of input device
CHNR	Integer	Write	Choice number that is the current measure of the choice device

Constants

Defined Argument	Constant	Description
STAT	GOK	Input obtained
	GNCHOI	Sampled without choosing

Description

The SAMPLE CHOICE function writes the current measure of the specified choice-class logical input device to the corresponding output argument.

If the input is valid, the function writes 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 NOCHOICE, and if the user did not move the prompt from its initial position, this function writes NOCHOICE to the status argument. This indicates that the user has not yet made a choice.

See Also

SET CHOICE MODE

SAMPLE LOCATOR

Operating States

WSOP, WSAC, SGOP

Syntax

GSMLC (WKID, DEVNUM, TNR, LPX, LPY)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Locator device number
TNR	Integer	Write	Normalization transformation number
LPX, LPY	Real	Write	Locator position expressed as a WC point

Description

The SAMPLE LOCATOR function writes the current measure of the specified locator-class logical input device and the corresponding normalization transformation number to the appropriate output arguments.

See Also

SET LOCATOR MODE

SAMPLE LOCATOR 3

Operating States

WSOP, WSAC, SGOP

Syntax

GSMLC3 (WKID, DEVNUM, TNR, VIEWI, PX, PY, PZ)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Locator device number
TNR	Integer	Write	Normalization transformation number
VIEWI	Integer	Write	View index
PX, PY, PZ	Real	Write	Locator position expressed as a WC point

Description

The SAMPLE LOCATOR 3 function writes the current measure of the specified locator-class logical input device and the corresponding normalization transformation number to the appropriate output arguments.

The SAMPLE LOCATOR 3 function returns the view index of the view mapping transformation last used to translate the NPC points to NDC points. See the *DEC GKS User's Guide* for more information on view indexes.

See Also

SET LOCATOR MODE

SAMPLE PICK

Operating States

WSOP, WSAC, SGOP

Syntax

GSMPK (WKID, DEVNUM, STAT, SGNA, PKID)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Pick device number
STAT	Integer (constant)	Write	Status of input device
SGNA	Integer	Write	Segment name
PKID	Integer	Write	Pick identifier

Constants

Defined Argument	Constant	Description
STAT	GOK	Input obtained
	GNPICK	Sampled without picking

Description

The SAMPLE PICK function writes the current measure of the specified pick-class logical input device to the corresponding output argument. This function writes OK to the status argument and writes the positive integers representing the picked segment and the pick identifier to the output arguments if the input is valid.

If the initial choice status is NOPICK, and if the user did not move the prompt, this function writes NOPICK to the status argument. This indicates that the user did not pick a segment yet. The logical input device also returns NOPICK if the user moved the prompt but the aperture is not touching a segment at the time of the sample.

See Also

SET PICK MODE
Example 9–2 for a program exampe using the SAMPLE PICK function

SAMPLE STRING

Operating States

WSOP, WSAC, SGOP

Syntax

GSMST (WKID, DEVNUM, LOSTR, STR)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	String device number
LOSTR	Integer	Write	Number of characters returned
STR	Character*(*)	Write	String returned

Description

The SAMPLE STRING function writes the current measure of the specified string-class logical input device to the appropriate output arguments.

When you call this function, the following two buffers exist:

- The application's string buffer, which you allocate before the call. You must specify the size to be at least one byte larger than the maximum possible string size. It is best to specify a string buffer size of 256 characters.
- The logical input device's string buffer, whose size you can specify in the call to the INITIALIZE STRING function.

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 sampled string in the application's buffer.
- Leaves any remaining characters in the device's buffer. You need to call this function again to access the remaining characters.

See Also

INITIALIZE STRING
SET STRING MODE

SAMPLE STRING (FORTRAN–77 Subset)

Operating States

WSOP, WSAC, SGOP

Syntax

GSMST (WKID, DEVNUM, LOSTR, STR)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	String device number
LOSTR	Integer	Write	Number of characters returned
STR	Character*80	Write	String returned

Description

The SAMPLE STRING function writes the current measure of the specified string-class logical input device to the appropriate output arguments.

When you call this function, the following two buffers exist:

- The application's string buffer, which you allocate before the call. You must specify the size to be at least one byte larger than the maximum possible string size. It is best to specify a string buffer size of 256 characters.
- The logical input device's string buffer, whose size you can specify in the call to the INITIALIZE STRING function.

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 sampled string in the application's buffer.
- Leaves any remaining characters in the device's buffer. You need to call this function again to access the remaining characters.

See Also

INITIALIZE STRING
SET STRING MODE

SAMPLE STROKE

Operating States

WSOP, WSAC, SGOP

Syntax

GSMSK (WKID, DEVNUM, N, TNR, NP, PXA, PYA)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Stroke device number
N	Integer	Read	Number of elements in each of the arrays for the stroke point coordinates
TNR	Integer	Write	Normalization transformation number
NP	Integer	Write	Number of points in the stroke
PXA(N), PYA(N)	Array of reals	Write	WC points in the stroke

Description

The SAMPLE STROKE function writes the current measure of the specified stroke-class logical input device to the corresponding output arguments.

When you call this function, the following two buffers exist:

- The application's stroke buffer, which you must allocate before the call. You must specify the buffer size in the *N* argument.
- The logical input device's stroke buffer, whose size you can specify in the call to the INITIALIZE STROKE function.

When sampling stroke input, DEC GKS accepts any initial stroke points and translates them 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 sampled stroke in the application's buffer.
- Leaves any remaining points in the device's buffer. You need to call this function again to access the remaining stroke points.

SAMPLE STROKE

See Also

INITIALIZE STROKE
SET STROKE MODE

SAMPLE STROKE 3

Operating States

WSOP, WSAC, SGOP

Syntax

GSMSK3 (WKID, DEVNUM, N, TNR, VIEWI, NP, SX, SY, SZ)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Stroke device number
N	Integer	Read	Number of elements in each of the arrays for the stroke point coordinates
TNR	Integer	Write	Normalization transformation number
VIEWI	Integer	Write	View index
NP	Integer	Write	Number of points returned
SX(N), SY(N), SZ(N)	Array of reals	Write	WC points in the stroke

Description

The SAMPLE STROKE 3 function writes the current measure of the specified stroke-class logical input device to the corresponding output arguments. The measure consists of a sequence of WC points, the normalization transformation number used in the conversion to WC points, and the view index used to convert the NPC points to NDC points.

When you call this function, the following two buffers exist:

- The application's stroke buffer, which you must allocate before the call. You must specify the buffer size in the *N* argument.
- The logical input device's stroke buffer, whose size is specified in the call to the INITIALIZE STROKE 3 function.

When sampling stroke input, DEC GKS accepts any initial stroke points and translates them 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 sampled stroke in the application's buffer.
- Leaves any remaining points in the device's buffer. This function must be called again to access the remaining stroke points.

SAMPLE STROKE 3

See Also

INITIALIZE STROKE 3
SET STROKE MODE

SAMPLE VALUATOR

Operating States

WSOP, WSAC, SGOP

Syntax

GSMVL (WKID, DEVNUM, VAL)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Valuator device number
VAL	Real	Write	Current measure of the valuator device

Description

The SAMPLE VALUATOR function writes the current measure of the specified valuator-class logical input device to the corresponding output argument.

See Also

SET VALUATOR MODE
Example 9–4 for a program example using the SAMPLE VALUATOR function

SET CHOICE MODE

Operating States

WSOP, WSAC, SGOP

Syntax

GSCHM (WKID, DEVNUM, MODE, ESW)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Choice device number
MODE	Integer (constant)	Read	Operating mode, specifying method of input
ESW	Integer (constant)	Read	Echo flag for prompt and input values

Constants

Defined Argument	Constant	Description
MODE	GREQU	Request mode. This is the default value.
	GSAMPL	Sample mode.
	GEVENT	Event mode.
ESW	GNECHO	Echo disabled.
	GECHO	Echo enabled. This is the default value.

Description

The SET CHOICE MODE function sets the specified choice device to the specified operating mode and sets the echo state of the device as specified. Depending on the input operating mode, an interaction with the device may begin or end.

The input device state defined by the operating mode and the echo switch are stored in the workstation state list for the specified choice device.

See Also

INITIALIZE CHOICE
Example 9–1 for a program example using a SET . . . MODE function

SET LOCATOR MODE

Operating States

WSOP, WSAC, SGOP

Syntax

GSLCM (WKID, DEVNUM, MODE, ESW)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Locator device number
MODE	Integer (constant)	Read	Operating mode, specifying method of input
ESW	Integer (constant)	Read	Echo flag for prompt and input values

Constants

Defined Argument	Constant	Description
MODE	GREQU	Request mode. This is the default value.
	GSAMPL	Sample mode.
	GEVENT	Event mode.
ESW	GNECHO	Echo disabled.
	GECHO	Echo enabled. This is the default value.

Description

The SET LOCATOR MODE function sets the specified locator device to the specified operating mode and sets the echo state of the device as specified. Depending on the input operating mode, an interaction with the device may begin or end.

The input device state defined by the operating mode and the echo switch are stored in the workstation state list for the specified locator device.

See Also

INITIALIZE LOCATOR
Example 9–1 for a program example using the SET LOCATOR MODE function

SET PICK MODE

Operating States

WSOP, WSAC, SGOP

Syntax

GSPKM (WKID, DEVNUM, MODE, ESW)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Pick device number
MODE	Integer (constant)	Read	Operating mode, specifying method of input
ESW	Integer (constant)	Read	Echo flag for prompt and input values

Constants

Defined Argument	Constant	Description
MODE	GREQU	Request mode. This is the default value.
	GSAMPL	Sample mode.
	GEVENT	Event mode.
ESW	GNECHO	Echo disabled.
	GECHO	Echo enabled. This is the default value.

Description

The SET PICK MODE function sets the specified pick device to the specified operating mode and sets the echo state of the device as specified. Depending on the input operating mode, an interaction with the device may begin or end.

The input device state defined by the operating mode and the echo switch are stored in the workstation state list for the specified pick device.

See Also

INITIALIZE PICK
Example 9–2 for a program example using the SET PICK MODE function

SET STRING MODE

Operating States

WSOP, WSAC, SGOP

Syntax

GSSTM (WKID, DEVNUM, MODE, ESW)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	String device number
MODE	Integer (constant)	Read	Operating mode, specifying method of input
ESW	Integer (constant)	Read	Echo flag for prompt and input values

Constants

Defined Argument	Constant	Description
MODE	GREQU	Request mode. This is the default value.
	GSAMPL	Sample mode.
	GEVENT	Event mode.
ESW	GNECHO	Echo disabled.
	GECHO	Echo enabled. This is the default value.

Description

The SET STRING MODE function sets the specified string device to the specified operating mode and sets the echo state of the device as specified. Depending on the input operating mode, an interaction with the device may begin or end.

The input device state defined by the operating mode and the echo switch are stored in the workstation state list for the specified string device.

See Also

INITIALIZE STRING
Example 9–1 for a program example using a SET . . . MODE function

SET STROKE MODE

Operating States

WSOP, WSAC, SGOP

Syntax

GSSKM (WKID, DEVNUM, MODE, ESW)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Stroke device number
MODE	Integer (constant)	Read	Operating mode, specifying method of input
ESW	Integer (constant)	Read	Echo flag for prompt and input values

Constants

Defined Argument	Constant	Description
MODE	GREQU	Request mode. This is the default value.
	GSAMPL	Sample mode.
	GEVENT	Event mode.
ESW	GNECHO	Echo disabled.
	GECHO	Echo enabled. This is the default value.

Description

The SET STROKE MODE function sets the specified stroke device to the specified operating mode and sets the echo state of the device as specified. Depending on the input operating mode, an interaction with the device may begin or end.

The input device state defined by the operating mode and the echo switch are stored in the workstation state list for the specified stroke device.

See Also

INITIALIZE STROKE
Example 9–1 for a program example using a SET . . . MODE function

SET VALUATOR MODE

Operating States

WSOP, WSAC, SGOP

Syntax

GSVLM (WKID, DEVNUM, MODE, ESW)

Argument	Data Type	Access	Description
WKID	Integer	Read	Workstation identifier
DEVNUM	Integer	Read	Valuator device number
MODE	Integer (constant)	Read	Operating mode, specifying method of input
ESW	Integer (constant)	Read	Echo flag for prompt and input values

Constants

Defined Argument	Constant	Description
MODE	GREQU	Request mode. This is the default value.
	GSAMPL	Sample mode.
	GEVENT	Event mode.
ESW	GNECHO	Echo disabled.
	GECHO	Echo enabled. This is the default value.

Description

The SET VALUATOR MODE function sets the specified valuator device to the specified operating mode and sets the echo state of the device as specified. Depending on the input operating mode, an interaction with the device may begin or end.

The input device state defined by the operating mode and the echo switch are stored in the workstation state list for the specified valuator device.

See Also

INITIALIZE VALUATOR
Example 9–4 for a program example using the SET VALUATOR MODE function

UNPACK DATA RECORD

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GUREC (LDR, DATREC, IIL, IRL, ISL, ERRIND, IL, IA, RL, RA, SL, LSTR, STR)

Argument	Data Type	Access	Description
LDR	Integer	Read	Number of elements used in *DATREC* array.
DATREC(LDR)	Character*80	Read	Data record.
IIL	Integer	Read	Dimension of integer array.
IRL	Integer	Read	Dimension of real array.
ISL	Integer	Read	Dimension of character array.
ERRIND	Integer	Write	Error indicator.
IL	Integer	Write	Number of integer entries.
IA(IIL)	Array of integers	Write	Array containing the integer entries.
RL	Integer	Write	Number of real entries.
RA(IRL)	Array of reals	Write	Array containing the real entries.
SL	Integer	Write	Number of string entries.
LSTR(ISL)	Array of integers	Write	Array containing the lengths of the character string entries.
STR(ISL)	Character*80	Write	Array containing the character string entries. Each element in the array should be dimension 80. For example, character*80 *STR(5)* is a 5-element array.

Description

The UNPACK DATA RECORD function unpacks a data record. This function does not generate an error. It reports error conditions through the *ERRIND* argument.

UNPACK DATA RECORD (FORTRAN–77 Subset)

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GUREC (LDR, DATREC, IIL, IRL, ISL, ERRIND, IL, IA, RL, RA, SL, LSTR, STR)

Argument	Data Type	Access	Description
LDR	Integer	Read	Number of elements used in *DATREC* array.
DATREC(LDR)	Character*80	Read	Data record.
IIL	Integer	Read	Dimension of integer array.
IRL	Integer	Read	Dimension of real array.
ISL	Integer	Read	Dimension of character array.
ERRIND	Integer	Write	Error indicator.
IL	Integer	Write	Number of integer entries.
IA(IIL)	Array of integers	Write	Array containing the integer entries.
RL	Integer	Write	Number of real entries.
RA(IRL)	Array of reals	Write	Array containing the real entries.
SL	Integer	Write	Number of string entries.
LSTR(ISL)	Array of integers	Write	Array containing the lengths of the character string entries.
STR(ISL)	Character*80	Write	Array containing the character string entries. Each element in the array should be dimension 80. For example, character*80 *STR(5)* is a 5-element array.

Description

The UNPACK DATA RECORD function unpacks a data record. This function does not generate an error. It reports error conditions through the *ERRIND* argument.

9.9 Program Examples

Example 9–1 illustrates the use of a locator-class logical input device in event mode. The program places a tracking plus sign (+) on the screen.

Example 9–1 Using a Locator-Class Logical Input Device in Event Mode

```
C  This program initializes and generates locator events.  Some of
C  the calls it uses include:  INQUIRE LOCATOR STATE,
C  SET LOCATOR MODE, and INITIALIZE LOCATOR.

      IMPLICIT NONE
      INCLUDE 'gks.f'

C  DATREC is a dummy argument.  The device handler ignores the
C  data record for all supported locator prompt and echo types.

C  ECHO_AREA is an array of real numbers that represent the
C  rectangular echo area, in device coordinates.  The echo area
C  defines a part of the workstation surface from which GKS
C  accepts input from the input prompt.

      INTEGER       CLASS
      CHARACTER*80  DATREC(1)
      INTEGER       DEVNUM
      REAL          ECHO_AREA (4)
      INTEGER       ERRIND
      INTEGER       ECHO_FLAG
      INTEGER       MODE
      INTEGER       PR_ECHO_TYPE
      INTEGER       REC_NUM
      INTEGER       REC_SIZE
      REAL          WRLD_X
      REAL          WRLD_Y
      INTEGER       WS_ID
      INTEGER       XFORM
      REAL          XMAX
      REAL          XMIN
      REAL          YMAX
      REAL          YMIN

C  Open the graphics environment:  open GKS, open the
C  workstation, activate the workstation, and set deferral state.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)
      CALL GSDS (WS_ID, GASAP, GALLOW)

C  Use INQUIRE LOCATOR DEVICE STATE to initialize the variables
C  you need to pass to the input functions.
C
C  GREALI tells the graphics handler to return the input
C  values as they are implemented.  (Use GSET to return the
C  values the way the application set them.)
C
C  After the function call, REC_SIZE contains the amount of the
C  buffer filled with the written data record.  If REC_NUM is
C  larger than REC_SIZE, GKS truncated the data record to fit
C  into the declared buffer.
```

(continued on next page)

Example 9–1 (Cont.) Using a Locator-Class Logical Input Device in Event Mode

```
      DEVNUM   = 1
      REC_SIZE = 10

      CALL GQLCS (WS_ID, DEVNUM, GREALI, REC_SIZE, ERRIND, MODE,
     *ECHO_FLAG, XFORM, WRLD_X, WRLD_Y, PR_ECHO_TYPE, ECHO_AREA,
     *REC_NUM, DATREC)

C  Check to see if the error status equals 0.

      IF (ERRIND .NE. 0) THEN
        CALL GMSG (WS_ID, 'ERRIND is not 0.')
        GOTO 10
      END IF

C  Set the initial position of the input prompt.

      WRLD_X = 0.9
      WRLD_Y = 0.0

C  Initialize the logical input device.

      XMIN = ECHO_AREA(1)
      XMAX = ECHO_AREA(2)
      YMIN = ECHO_AREA(3)
      YMAX = ECHO_AREA(4)

      CALL GINLC (WS_ID, DEVNUM, XFORM, WRLD_X, WRLD_Y,
     *PR_ECHO_TYPE, XMIN, XMAX, YMIN, YMAX, REC_NUM, DATREC)

C  Activate the logical input device by placing it in event mode.

      CALL GSLCM (WS_ID, DEVNUM, GEVENT, GECHO)

C  Instruct the user

      CALL GSCHH (0.03)
      CALL GTX (0.05, 0.95, 'Move the input prompt upwards.')
      CALL GTX (0.05, 0.90, 'Trigger until I say when to stop.')

C  Do until the use moves the input prompt closest to the top of the
C  workstation surface.
C
C  In the WHILE loop, the call to AWAIT EVENT immediately checks
C  the input queue (as specified by the timeout argument of 0.0).
C  If the user has not yet entered an event or if the application
C  has removed all reports generated so far, AWAIT EVENT returns
C  GNCLAS to its CLASS argument.
C
C  This program uses only the locator input class to generate
C  events.  The IF statement keeps calling GET LOCATOR as long as
C  the input_class argument is GLOCAT.  The program stops calling
C  GET LOCATOR when the CLASS argument becomes GNCLAS.

      DO WHILE (WRLD_Y .LT. 0.9)
```

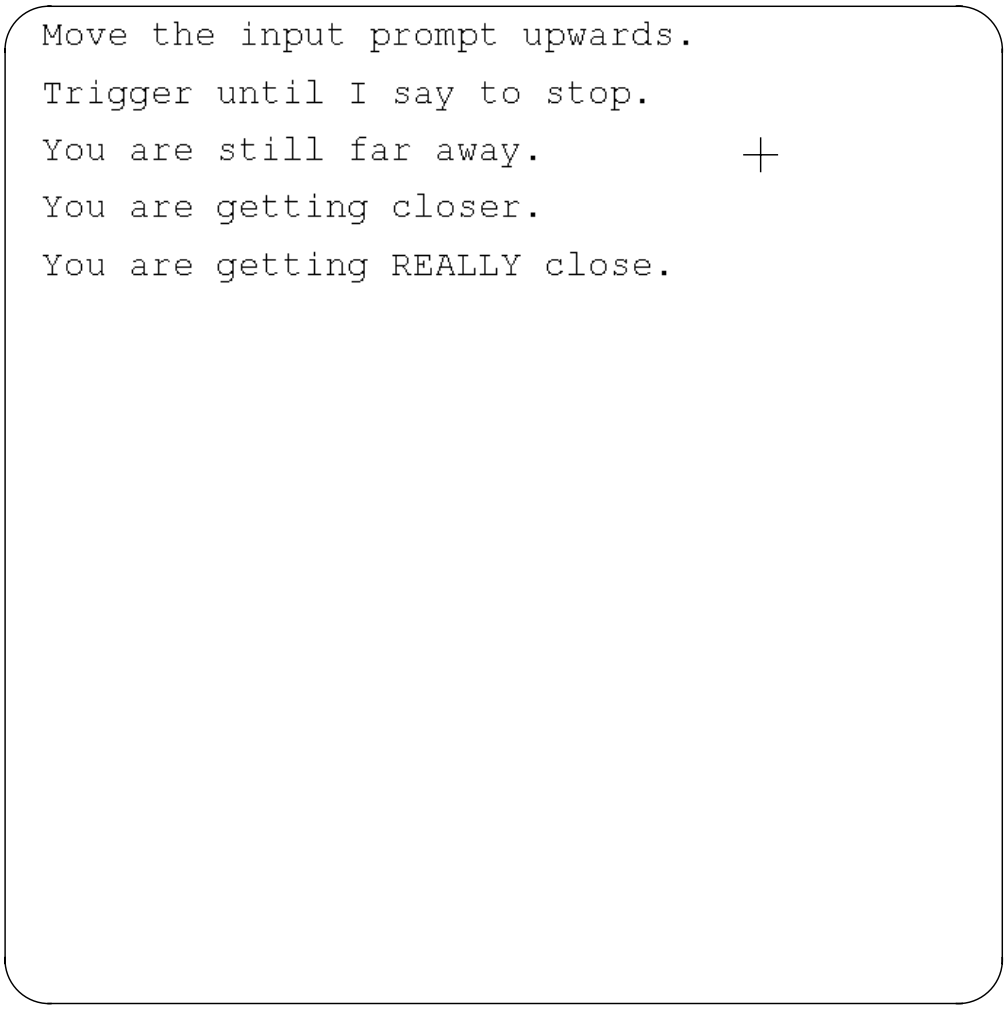
(continued on next page)

Example 9–1 (Cont.) Using a Locator-Class Logical Input Device in Event Mode

```
C  Check the event queue.

      CALL GWAIT (0.0, WS_ID, CLASS, DEVNUM)

      IF (CLASS .NE. GNCLAS) THEN
         CALL GGTLC (XFORM, WRLD_X, WRLD_Y)
      ENDIF

C  Tease the user as the input prompt gets closer.

      IF ((WRLD_Y .GT. 0.1) .AND.
     *     (WRLD_Y .LT. 0.5)) THEN
           CALL GTX (0.05, 0.85, 'You are still far away.')
      ENDIF

      IF ((WRLD_Y .GT. 0.5) .AND.
     *     (WRLD_Y .LT. 0.7)) THEN
           CALL GTX (0.05, 0.80, 'You are getting closer.')
      ENDIF
      IF ((WRLD_Y .GT. 0.7) .AND.
     *     (WRLD_Y .LT. 0.9)) THEN
           CALL GTX (0.05, 0.75, 'You are REALLY close.')
      ENDIF
      ENDDO

      CALL GTX (0.05, 0.70, 'YOU MADE IT!!!')

C  Deactivate the logical input device by placing it in request mode.
C  (You only have to return to request mode, though, if you are going
C  to use a different type of input mode besides event later in
C  the program.)

      CALL GSLCM (WS_ID, DEVNUM, GREQU, GECHO)

   10 CONTINUE

C  Deactivate the workstation, close the workstation, and close GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 9–2 shows a workstation screen after the user has moved the input prompt near the top of the screen.

Figure 9–2 Input Prompt Near the Top of the Screen



Example 9–2 illustrates the use of the SAMPLE PICK function.

Example 9–2 Using a Pick-Class Logical Input Device in Sample Mode

```
C  This program initializes and samples pick input.  Some of the
C  calls include:  SET FILL INTERIOR STYLE, SET PICK ID,
C  SET FILL AREA COLOR INDEX, FILL AREA, CREATE SEGMENT, CLOSE SEGMENT,
C  SET TEXT HEIGHT, TEXT, INQUIRE PICK DEVICE STATE, INITIALIZE PICK,
C  SET PICK MODE, and SAMPLE PICK.

      IMPLICIT NONE
      INCLUDE 'gks.f'
```

(continued on next page)

Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode

```
C  ECHO_AREA is an array of real numbers that represent the
C  rectangular echo area, in device coordinates.  The echo area
C  defines the workstation surface from which GKS accepts input
C  from the input prompt.

      INTEGER      BOX_1
      INTEGER      BOX_2
      CHARACTER*80 DATREC(1)
      INTEGER      DEVNUM
      REAL         ECHO_AREA (4)
      INTEGER      ECHO_FLAG
      INTEGER      ERR_IND
      INTEGER      INIT_STATUS
      INTEGER      IN_MODE
      INTEGER      IN_STATUS
      INTEGER      NUM_POINTS
      INTEGER      PICK_ID
      INTEGER      PR_ECHO_TYPE
      INTEGER      REC_LEN
      INTEGER      REC_SIZE
      INTEGER      SEGMENT
      INTEGER      TRI_1
      INTEGER      TRI_2
      INTEGER      WS_ID
      REAL         XMAX
      REAL         XMIN
      REAL         X_VALUES (4)
      REAL         YMAX
      REAL         YMIN
      REAL         Y_VALUES (4)

C  Open the graphics environment:  open GKS, open the
C  workstation, activate the workstation, and set the deferral state.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)
      CALL GSDS (WS_ID, GASAP, GALLOW)

C  Create the divided boxes.

      CALL GSFAIS (GSOLID)

C  Establish the position of the first box.

      X_VALUES(1) = 0.1
      X_VALUES(2) = 0.4
      X_VALUES(3) = 0.1
      X_VALUES(4) = 0.1

      Y_VALUES(1) = 0.3
      Y_VALUES(2) = 0.6
      Y_VALUES(3) = 0.6
      Y_VALUES(4) = 0.3

C  Create the box on the left side of the workstation surface
C  and place it in a segment.  Divide the box diagonally and
C  set pick identifiers for each of the created triangles.
```

(continued on next page)

Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode

```
      BOX_1 = 1
      NUM_POINTS = 4
      TRI_1 = 1
      TRI_2 = 2

      CALL GCRSG (BOX_1)
      CALL GSPKID (TRI_1)
      CALL GSFACI (2)
      CALL GFA (NUM_POINTS, X_VALUES, Y_VALUES)

      X_VALUES(3) = 0.4
      Y_VALUES(3) = 0.3

      CALL GSPKID (TRI_2)
      CALL GSFACI (3)
      CALL GFA (NUM_POINTS, X_VALUES, Y_VALUES)
      CALL GCLSG ()

C  Reset the X and Y world coordinate values to change the
C  position of the box.

      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

C  Create the box on the right side of the workstation surface
C  and place it in a segment.  Divide the box diagonally and
C  set pick identifiers for each of the created triangles.

      BOX_2 = 2

      CALL GSPKID (TRI_1)
      CALL GCRSG (BOX_2)
      CALL GSFACI (2)
      CALL GFA (NUM_POINTS, X_VALUES, Y_VALUES)

      X_VALUES (3) = 0.9
      Y_VALUES (3) = 0.3

      CALL GSPKID (TRI_2)
      CALL GSFACI (3)
      CALL GFA (NUM_POINTS, X_VALUES, Y_VALUES)
      CALL GCLSG ()
      CALL GSDTEC (BOX_1, GDETEC)
      CALL GSDTEC (BOX_2, GDETEC)

C  Label the triangles by their pick identifiers.

      CALL GSCHH (0.03)
      CALL GTX (0.2, 0.45, '1')
      CALL GTX (0.3, 0.45, '2')
      CALL GTX (0.7, 0.45, '1')
      CALL GTX (0.8, 0.45, '2')

C  Use INQUIRE PICK DEVICE STATE to initialize the variables
C  you need to pass to the input function.  GREALI tells the
C  graphics handler to pass the input values as they
C  are implemented.  (Use GSET to tell the application
C  to pass the input values the way the application set them.)
C
C  After the function call, REC_SIZE contains the amount of the
C  buffer filled with the written data record.  If REC_NUM is
C  larger than REC_SIZE, you know GKS truncated the data record
C  to fit into your declared buffer.
```

(continued on next page)

Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode

```
      DEVNUM   = 1
      REC_SIZE = 10

      CALL GQPKS (WS_ID, DEVNUM, GREALI, REC_SIZE, ERR_IND,
     *IN_MODE, ECHO_FLAG, INIT_STATUS, SEGMENT, PICK_ID,
     *PR_ECHO_TYPE, ECHO_AREA, REC_LEN, DATREC)

C  Check to see if the error status equals 0.

      IF (ERR_IND .NE. 0) THEN
        CALL GMSG (WS_ID, 'ERR_IND is not 0.')
        GOTO 10
      END IF

C  Use INITIALIZE PICK to initialize the logical input device.

C  Assign new values to the input variables.

      INIT_STATUS = GOK
      PICK_ID = TRI_1
      SEGMENT = BOX_1
      XMIN = ECHO_AREA(1)
      XMAX = ECHO_AREA(2)
      YMIN = ECHO_AREA(3)
      YMAX = ECHO_AREA(4)

      CALL GINPK (WS_ID, DEVNUM, INIT_STATUS, SEGMENT,
     *PICK_ID, PR_ECHO_TYPE, XMIN, XMAX, YMIN, YMAX,
     *REC_SIZE, DATREC)

C  Activate the logical input device by placing it in sample mode.
C  The input prompt now appears on the workstation surface and the
C  user can change the measure of the device.

      CALL GSPKM (WS_ID, DEVNUM, GSAMPL, GECHO)

C  Tell the user the task.

      CALL GSCHH (0.03)
      CALL GTX (0.05, 0.95, 'Move the cursor to a triangle.')
      CALL GTX (0.05, 0.90, 'I will say if it is correct.')

C  Retrieve the current input value without the user having to
C  trigger the device by using SAMPLE PICK.  Use WHILE to do it
C  until the user picks the second triangle in the second box.

      DO WHILE ((SEGMENT .NE. 2) .OR. (PICK_ID .NE. 2))

      CALL GSMPK (WS_ID, DEVNUM, IN_STATUS,
     *SEGMENT, PICK_ID)

C  Tease the user.

      IF ((SEGMENT .EQ. 1) .AND. (PICK_ID .EQ. 1)) THEN
         CALL GTX (0.05, 0.85,
     *               'You are pretty far away.')
      ENDIF
      IF ((SEGMENT .EQ. 1) .AND. (PICK_ID .EQ. 2)) THEN
         CALL GTX (0.05, 0.80,
     *               'You are getting closer.')
      ENDIF
      IF ((SEGMENT .EQ. 2) .AND. (PICK_ID .EQ. 1)) THEN
         CALL GTX (0.05, 0.75,
     *               'You are REALLY close.')
      ENDIF
      ENDDO
```

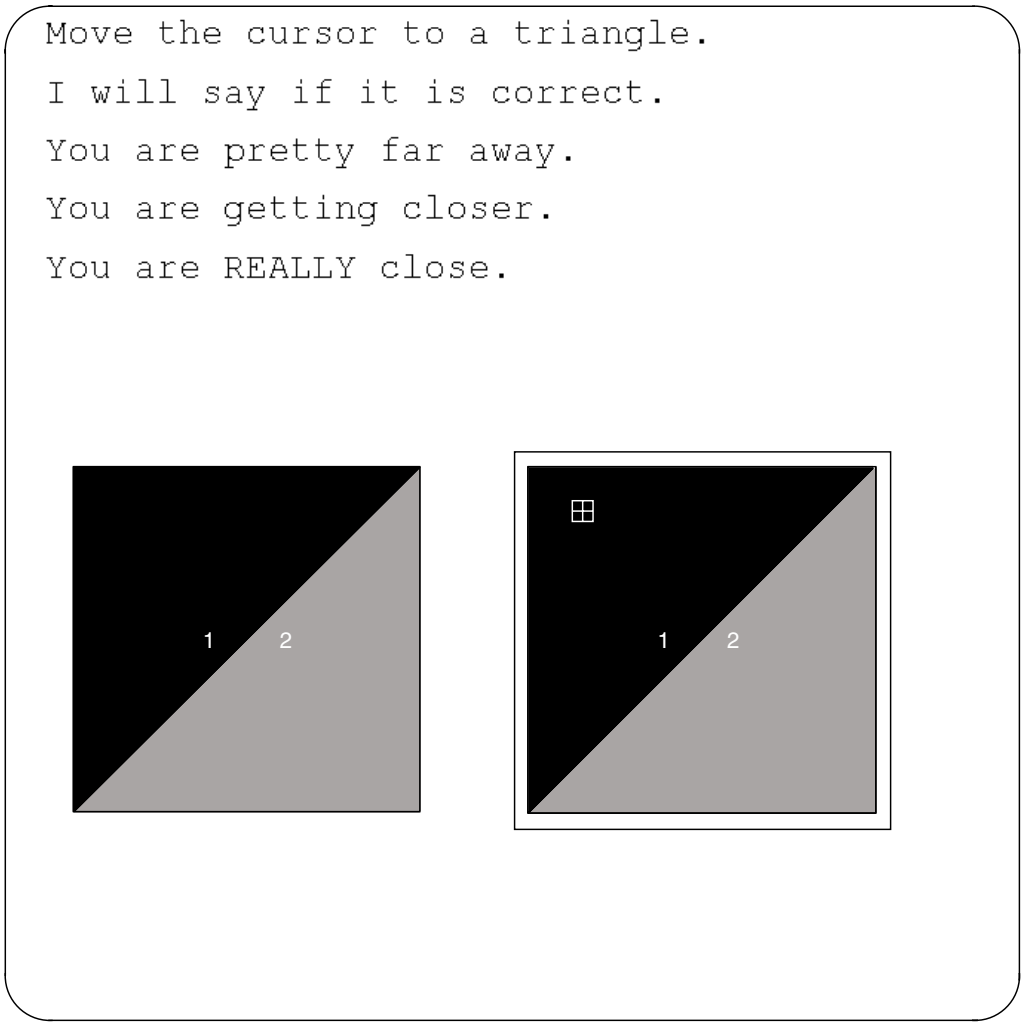
(continued on next page)

Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode

```
      CALL GTX (0.05, 0.70, 'YOU MADE IT!!!')

C  Deactivate the logical input device by placing it in request
C  mode.  The device handler removes the aperture from the
C  workstation surface and the user can no longer enter input.

      CALL GSPKM (WS_ID, DEVNUM, GREQU, GECHO)

C  Deactivate the workstation, close the workstation, and
C  close GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()
      END
```

Figure 9–3 shows the workstation surface when the user picks the correct triangle.

Figure 9–3 Picking the Correct Triangle



ZK–4077A–GE

Example 9–3 illustrates the use of the INITIALIZE STRING function.

Example 9–3 Using a String-Class Logical Input Device in Request Mode

```
C  This program initializes and requests string input.  Some of the
C  calls it uses include:  INQUIRE STRING STATE, INITIALIZE STRING,
C  SET STRING MODE, and REQUEST STRING.

      IMPLICIT NONE
      INCLUDE 'gks.f'
```

(continued on next page)

Example 9–3 (Cont.) Using a String-Class Logical Input Device in Request Mode

```
C  The data record contains the buffer length and the initial editing
C  position for all logical input prompt and echo types.  The buffer
C  can be only as long as the maximum size the workstation supports.
C  To obtain the maximum buffer size, call INQUIRE DEFAULT STRING
C  DEVICE DATA.
C
C  ECHO_AREA is an array of real numbers that represents the
C  rectangular echo area, in device coordinates.  The echo area
C  defines the workstation surface from which GKS accepts input
C  from the input prompt.
C
C  The defined string variables contain a string that is the width of
C  the terminal screen.  You can change the maximum size of the input
C  string every time you initialize the string logical input device by
C  changing the value associated with the buffer length.

      INTEGER      CURSOR_POS
      CHARACTER*80 DATREC(10)
      CHARACTER*80 DEFAULT_STRING
      INTEGER      DEVNUM
      REAL         ECHO_AREA (4)
      INTEGER      ECHO_FLAG
      INTEGER      ERR_IND
      INTEGER      IN_MODE
      INTEGER      IN_STATUS
      INTEGER      PR_ECHO_TYPE
      INTEGER      REC_BUF_LEN
      INTEGER      REC_NUM
      INTEGER      REC_SIZE
      INTEGER      RET_STRING_SIZE
      INTEGER      WS_ID
      REAL         XMAX
      REAL         XMIN
      REAL         YMAX
      REAL         YMIN

C  Open the graphics environment:  open GKS, open the
C  workstation, activate the workstation, and set deferral state.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)
      CALL GSDS (WS_ID, GASAP, GALLOW)

C  Use INQUIRE STRING DEVICE STATE to initialize the variables
C  you need to pass to the input functions.

      DEVNUM      = 1
      REC_BUF_LEN = 15
      REC_SIZE    = 10

      CALL GQSTS (WS_ID, DEVNUM, REC_SIZE, ERR_IND, IN_MODE,
     *ECHO_FLAG, RET_STRING_SIZE, DEFAULT_STRING, PR_ECHO_TYPE,
     *ECHO_AREA, REC_BUF_LEN, CURSOR_POS, REC_NUM, DATREC)
```

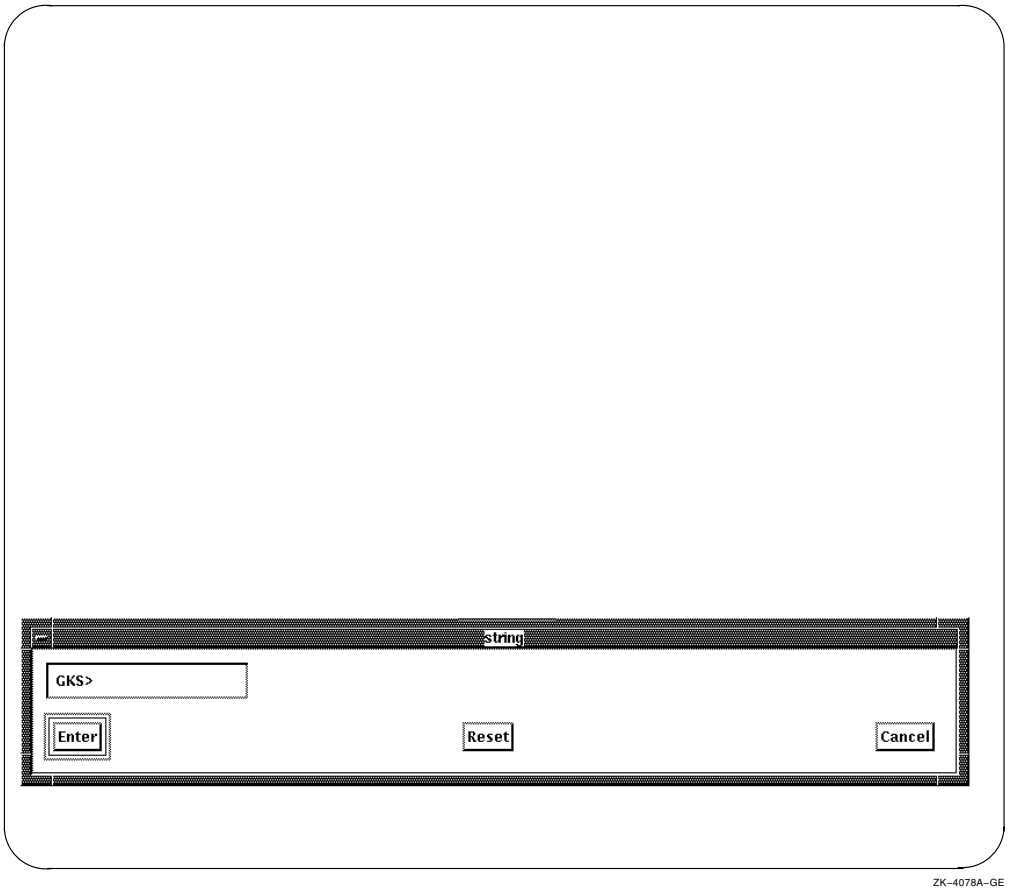
(continued on next page)

Example 9–3 (Cont.) Using a String-Class Logical Input Device in Request Mode

```
C  Check to see if the error status equals 0

      IF (ERR_IND .NE. 0) THEN
        CALL GMSG (WS_ID, 'ERR_IND is not 0.')
        GOTO 10
      END IF

C  Assign new values to the input variables.

      PR_ECHO_TYPE  = 1
      REC_BUF_LEN   = 15

C  Initialize the logical input device.

      RET_STRING_SIZE = 4
      XMIN = ECHO_AREA(1)
      XMAX = ECHO_AREA(2)
      YMIN = ECHO_AREA(3)
      YMAX = ECHO_AREA(4)

      CALL GINST (WS_ID, DEVNUM, RET_STRING_SIZE, 'GKS>',
     *PR_ECHO_TYPE, XMIN, XMAX, YMIN, YMAX,
     *REC_BUF_LEN, CURSOR_POS, REC_NUM, DATREC)

C  Activate the logical input device by calling REQUEST STRING.
C  GKS writes the string to the next-to-last argument.  The last
C  argument contains the size of the input string.

      CALL GRQST (WS_ID, DEVNUM, IN_STATUS, RET_STRING_SIZE,
     *DEFAULT_STRING)

10    CONTINUE

C  Deactivate the workstation, close the workstation, and close GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()

C  Output the input string and its size.

      WRITE (6,*) DEFAULT_STRING, RET_STRING_SIZE, IN_STATUS

      END
```

Figure 9–4 shows a workstation screen at the request for input.

Figure 9–4 Requesting Input from a String-Class Logical Input Device in Request Mode



Example 9–4 illustrates the use of the SAMPLE VALUATOR function.

Example 9–4 Using a Valuator-Class Logical Input Device in Sample Mode

```
C  This program initializes and samples valuator input.  Some of
C  the calls it uses include:  INQUIRE VALUATOR STATE, INITIALIZE
C  VALUATOR, SET VALUATOR MODE, SAMPLE VALUATOR, EVALUATE
C  TRANSFORMATION MATRIX, and SET SEGMENT TRANSFORMATION.

      IMPLICIT NONE
      INCLUDE 'gks.f'
```

(continued on next page)

Example 9–4 (Cont.) Using a Valuator-Class Logical Input Device in Sample Mode

```
C  ECHO_AREA is an array of real numbers that represent the
C  rectangular echo area, in device coordinates.  The echo area
C  defines the workstation surface from which GKS accepts input
C  from the input prompt.
C
C  The graphics handler uses two parts of the valuator input data
C  record for prompt and echo type 1:  the real value representing
C  an upper limit and another real value representing a lower limit.
C
C  Your terminal might support one of three valuator prompt and echo
C  types represented by the integers 1, 2, and 3.  Types 1 and 2
C  prompt the user with a rectangle and a horizontal scale.  To use
C  the first two types, the user uses the arrow keys or the mouse to
C  move an arrow along the scale between the upper and lower limits.
C  With type 3, GKS changes a single digital representation of the real
C  values between the upper and lower limits.  The user controls the change
C  of numbers with the arrow keys or the mouse.

      INTEGER      BOX
      REAL         BOX_X (5)
      REAL         BOX_Y (5)
      CHARACTER*80 DATREC(10)
      INTEGER      DEVNUM
      REAL         ECHO_AREA (4)
      INTEGER      ECHO_FLAG
      INTEGER      ERR_IND
      REAL         HIGH_VALUE
      REAL         INIT_VALUE
      INTEGER      IN_MODE
      REAL         LARGER
      REAL         LOW_VALUE
      INTEGER      PR_ECHO_TYPE
      INTEGER      REC_NUM
      INTEGER      REC_SIZE
      INTEGER      WS_ID
      REAL         XFORM_MATRIX (6)
      REAL         XMAX
      REAL         XMIN
      REAL         YMAX
      REAL         YMIN

C  Open the graphics environment:  open GKS, open the
C  workstation, activate the workstation, and set deferral state.

      WS_ID = 1

      CALL GOPKS (6, 0)
      CALL GOPWK (WS_ID, GCONID, GWSDEF)
      CALL GACWK (WS_ID)
      CALL GSDS (WS_ID, GASAP, GALLOW)

C  Use INQUIRE VALUATOR DEVICE STATE to initialize the variables
C  you need to pass to the input functions.
C
C  After the function call, REC_SIZE contains the amount of the
C  buffer filled with the written data record.  If REC_NUM is
C  larger than REC_SIZE, GKS truncated the data record to fit into
C  the declared buffer.

      DEVNUM   = 1
      REC_SIZE = 10
```

(continued on next page)

Example 9–4 (Cont.) Using a Valuator-Class Logical Input Device in Sample Mode

```
      CALL GQVLS (WS_ID, DEVNUM, REC_SIZE, ERR_IND, IN_MODE,
     *ECHO_FLAG, INIT_VALUE, PR_ECHO_TYPE, ECHO_AREA,
     *LOW_VALUE, HIGH_VALUE, REC_NUM, DATREC)

C  Check to see if the error status equals 0

      IF (ERR_IND .NE. 0) THEN
        CALL GMSG(WS_ID, 'ERR_IND is not 0.')
        GOTO 10
      END IF

C  Assign new values to the input variables.

      HIGH_VALUE = 2.0
      INIT_VALUE = 1.0
      LOW_VALUE  = 0.001

C  Initialize the logical input device.

      XMIN = ECHO_AREA(1)
      XMAX = ECHO_AREA(2)
      YMIN = ECHO_AREA(3)
      YMAX = ECHO_AREA(4)

      CALL GINVL (WS_ID, DEVNUM, INIT_VALUE, PR_ECHO_TYPE,
     *XMIN, XMAX, YMIN, YMAX, LOW_VALUE, HIGH_VALUE,
     *REC_NUM, DATREC)

C  Activate the logical input device by placing it in sample mode.
C  The input prompt now appears on the workstation surface and
C  the user can change the measure of the device.

      CALL GSVLM (WS_ID, DEVNUM, GSAMPL, GECHO)

C  Create a box.  (The program will scale the box according to
C  the sampled value of the valuator device.)

      LARGER = 0.03

      CALL GSFAIS (GSOLID)
      CALL GSCHH (LARGER)

      BOX_X(1) = 0.4
      BOX_Y(1) = 0.4

      BOX_X(2) = 0.6
      BOX_Y(2) = 0.4

      BOX_X(3) = 0.6
      BOX_Y(3) = 0.6

      BOX_X(4) = 0.4
      BOX_Y(4) = 0.6

      BOX_X(5) = 0.4
      BOX_Y(5) = 0.4

      BOX = 2

      CALL GCRSG (BOX)
      CALL GFA (5, BOX_X, BOX_Y)
      CALL GCLSG ()
```

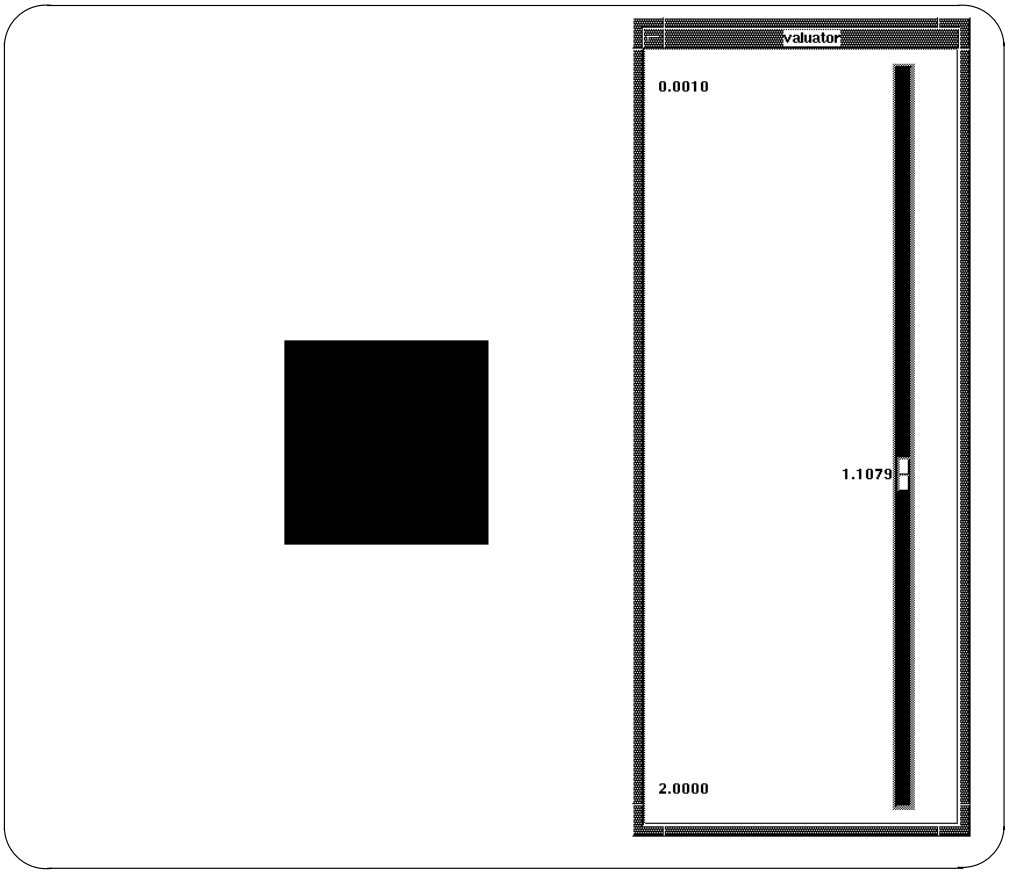
(continued on next page)

Example 9–4 (Cont.) Using a Valuator-Class Logical Input Device in Sample Mode

```
C  Display instructions to the user.

      CALL GTX (0.05, 0.95, 'Change the box''s size.')
      CALL GTX (0.05, 0.90, 'To stop, set the value to 2.0.')

C  Sample the user input by using SAMPLE VALUATOR.
C
C  The call to SAMPLE VALUATOR retrieves the current input
C  value (without the user having to trigger the logical input
C  device).  The WHILE loop ends when the user moves the prompt
C  to 2.0.

      DO WHILE (INIT_VALUE .NE. 2.0)

      CALL GSMVL (WS_ID, DEVNUM, INIT_VALUE)

C  Scale the segment according to the INIT_VALUE argument using
C  EVALUATE TRANSFORMATION MATRIX.

      CALL GEVTM (0.5, 0.5, 0.0, 0.0, 0.0,
     *INIT_VALUE, INIT_VALUE, GWC, XFORM_MATRIX)

      IF (INIT_VALUE .NE. 1.0) THEN
         CALL GSSGT (BOX, XFORM_MATRIX)
         CALL GUWK (WS_ID, GPERFO)
      ENDIF
      ENDDO

C  Deactivate the logical input device by placing it in request mode.
C  The input prompt now appears on the workstation surface and
C  the user can change the measure of the device.

      CALL GSVLM (WS_ID, DEVNUM, GREQU, GECHO)

10    CONTINUE

C  Deactivate the workstation, close the workstation, and close GKS.

      CALL GDAWK (WS_ID)
      CALL GCLWK (WS_ID)
      CALL GCLKS ()

C  Display the sampled value.

      WRITE (6,*), INIT_VALUE

      END
```

Figure 9–5 shows the workstation surface after DEC GKS activates the valuator-class logical input device in sample mode.

Figure 9–5 Workstation Surface after Activating a Valuator-Class Logical Input Device in Sample Mode



Metafile Functions

Insert tabbed divider here. Then discard this sheet.

10
Metafile Functions

The DEC GKS **metafile** functions provide a mechanism for long-term storage, communication, and reproduction of a graphic 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 GKS Metafiles (GKSM or GKS3) or Computer Graphics Metafiles (CGM). GKSM metafiles are two-dimensional metafiles and GKS3 metafiles are three-dimensional metafiles.

The GKSM format is defined by the GKS standard; the GKS3 format is defined by the GKS–3D standard. When using the GKSM or GKS3 format, DEC GKS stores an audit of the generation of DEC GKS primitives. For more information concerning GKSM and GKS3 format, see Section 10.1 and the metafile appendix in the *DEC GKS User's Guide*.

The CGM format 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 graphic picture. CGM format is designed for use with many types of graphics applications, including DEC GKS applications. If you need to create a CGM for use with other applications, possibly on other systems, you can use DEC GKS to create the file. However, DEC GKS cannot read CGM format. For more information concerning CGM, see Section 10.2.

A short-term method of storing output primitives is to store them in segments. For more information concerning segments, see Chapter 8, Segment Functions.

10.1 Creating a GKSM or GKS3 Metafile

To create a GKSM or GKS3, you open and then activate a metafile workstation using the constant GMOUTP (numeric value 2) as a workstation type for category GMO workstations. As the device connection, name 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 GMOUTP workstations as determined by the maximum allowable open and active workstations, sending output to the active GMOUTP workstation. Specify the file type values, GKSM or GKS3, with the appropriate environment option to indicate a two- or three-dimensional GKS metafile. For more information on these environment options, see Chapter 2 and Chapter 3.

Once the GMOUTP type 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. Category metafile output workstations record the following information:

- The control functions that affect the appearance of the picture on the workstation surface.
- Output primitives, if the GMOUTP 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 GMOUTP workstation is active at the time of the call to CREATE SEGMENT.
- Geometric attribute data (such as character height, character-up vector, and so on) affecting stored text primitives, in a form equivalent to NDC points.
- Normalization transformation information such as the clipping rectangle. DEC GKS does not record workstation transformations.
- Data specific to the application, or information that DEC GKS metafiles cannot store through standard calls to DEC GKS functions (stored using the function WRITE ITEM TO GKSM).

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. Because metafiles record information pertinent to output only, DEC GKS metafiles do not record information about input function calls.

When you create a GKSM or GKS3 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 metafile header contains information such as the author of the metafile, the date, the version number, and the length of the different fields in the data record. 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 a GKSM or GKS3 (metafile header followed by an item header followed by an item data record, and so on), but the individual item data record format is implementation specific. For example, some implementations may store all item data as a string of characters, whereas other 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 example, an item of type 3 corresponds to a call to UPDATE WORKSTATION. The item type is contained in the item header. For a list of the integer values, see the appendix on metafiles in *DEC GKS User's Guide*.

When creating a GKSM or GKS3 metafile, you do not need to know the information contained in the item header or the item data record. Once you activate a type GMOUTP workstation and call output functions, DEC GKS formats the graphic output information within the metafile for you.

When you close the GMOUTP 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 a CGM

To create a CGM, you open and then activate a workstation using the constant GCGMO (numeric value 7) as a workstation type for category GMO workstations. As the device connection, name 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 type GCGMO workstations as determined by the maximum allowable open and active workstations, sending appropriate output to the appropriate active type GCGMO workstation.

Once the GCGMO workstation is active, DEC GKS places the graphic 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.
Picture descriptor elements	Define the limits of the virtual device coordinates (VDC) and the parameter modes for the attribute elements.
Control elements	Specify size and precision of VDC points, and format descriptions of the CGM elements.
Graphic primitive elements	Describe the geometric objects in the picture.
Attribute elements	Describe the various appearances of the graphic elements.
Escape elements	Describe device- and system-specific functionality.
External elements	Pass information not needed for the creation of a picture (for example, a message sent to the user of the metafile).

The elements may have associated data. For example, the graphic 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 the CGM elements are grouped into structures 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 or GKS3 items, CGM elements have a certain format, or **encoding**. DEC GKS can create CGM elements in one of the following encodings:

Encoding	Description
Character	This encoding requires that the CGM elements and their parameters be stored in a character-coded format as specified by the CGM standard. With this encoding, your metafiles use a minimum amount of physical storage.
Binary	This encoding requires that the CGM elements and their parameters be stored in binary code. Using this encoding, many of the applications and machines can store and read a CGM with greater ease.
Clear text	This encoding requires that the CGM elements and their parameters be stored in text. Using this encoding, you can type, print, or edit the CGM so you can review its contents before reading the file.

You can use bit mask constant values within your program to specify an encoding. An example of such a FORTRAN binding call is as follows:

```
CGM_FILE = 10
OPEN (UNIT=CGM_FILE, FILE='cgm_file.txt')
WSTYPE = GCGMO .OR. GMCHAR
CALL GOPWK (WSID, CGM_FILE, WSTYPE)
```

All the available constant values are listed in the language-dependent header files. The *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* contains extended information concerning bitmasks.

When you create a CGM, you do not need to know the information contained in the individual elements. Once you activate a type GCGMO workstation and call output functions, DEC GKS formats the graphic output information within the metafile for you.

For detailed information concerning the CGM format for the supported encodings, see the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*.

10.3 Reading a GKSM or GKS3 Metafile

To reproduce a graphic image from a GKSM or GKS3, you must open a metafile input (GMI) workstation. DEC GKS defines the constant GMINPT (numeric value 3) as the workstation type for category GMI workstations. Also, when you open the type GMINPT 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 open only one type GMINPT workstation for every corresponding physical file. DEC GKS distinguishes between GKSM and GKS3 files directly from the contents of the files.

When you open a type GMINPT workstation, the first item written to the metafile becomes the **current item**. The current item is the item processed when you call the function GET ITEM TYPE FROM GKSM. You can open as many type GMINPT 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 the following functions for all the applicable items in the metafile, until you reach the item type 0 (signifying the last item):

- GET ITEM TYPE FROM GKSM—Returns the item type and the length of the data record of the current item.

- READ ITEM FROM GKSM—Returns the item data record and causes the next item in the metafile to become the current item.
- INTERPRET ITEM—Reads information about an item and reproduces the desired action on all active workstations of categories GOUTPT and GOUTIN.

In most applications, you call INTERPRET ITEM for all items in a metafile. However, there are cases when you may not wish to do this.

For example, if the creator of the metafile called the function WRITE ITEM TO GKSM to pass user-defined data to the metafile, you need to handle this information in a special manner. For example, if the user-defined data is a text string containing information for the application programmer, then instead of passing the record to 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 GKSM user-defined items. Item types less than 0 are GKS3 user-defined items. DEC GKS metafiles reserve item data numbers 0 to 100. If you are not using a DEC GKS GKSM or GKS3, 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 UPDATE WORKSTATION), 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 UPDATE WORKSTATION, see Chapter 4, Control Functions.

If after calling GET ITEM TYPE FROM GKSM, you decide that you do not want to interpret the item, pass the value 0 as the data length argument to READ ITEM FROM GKSM. This skips the current item, causing the next item in the file to become the current item.

10.4 Metafile Inquiries

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

INQUIRE LEVEL OF GKS
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES
INQUIRE OPERATING STATE VALUE
INQUIRE SET OF OPEN WORKSTATIONS
INQUIRE WORKSTATION STATE

For more information concerning device-independent programming, see the *DEC GKS User's Guide*.

10.5 Function Descriptions

This section describes the DEC GKS metafile functions in detail.

GET ITEM TYPE FROM GKSM

Operating States

WSOP, WSAC, SGOP

Syntax

GGTITM (WKID, TYPE, IDRL)

Argument	Data Type	Access	Description
WKID	Integer	Read	Open metafile input workstation identifier. More than one MI workstation can be open.
TYPE	Integer	Write	Metafile item type.
IDRL	Integer	Write	Length of the item data record, in bytes. This may be passed to the READ ITEM FROM GKSM function.

Description

The GET ITEM TYPE FROM GKSM function returns the item type and the length of the item data record from the current item in a metafile to the last argument.

See Also

OPEN WORKSTATION
READ ITEM FROM GKSM
WRITE ITEM TO GKSM

INTERPRET ITEM

Operating States

GKOP, WSOP, WSAC, SGOP

Syntax

GIITM (TYPE, IDRL, LDR, DATREC)

Argument	Data Type	Access	Description
TYPE	Integer	Read	Metafile item type. This value can be obtained by calling the GET ITEM TYPE FROM GKSM function.
IDRL	Integer	Read	Length (in bytes) of the item data record. This value is returned by the GET ITEM TYPE FROM GKSM function.
LDR	Integer	Read	Length (in bytes) of the data record array.
DATREC(LDR)	Character*80	Read	Item's data record. This value is returned by the READ ITEM FROM GKSM function.

Description

The INTERPRET ITEM function interprets an item data record obtained by a call to READ ITEM FROM GKSM.

If the item type corresponds to a call to a function that affects graphic representation, this function makes appropriate changes to the GKS state list, and generates the specified graphic output on all active workstations of categories OUTPUT and OUTIN.

If the item type identifies user-defined data, this function generates an error indicating that it cannot interpret the item.

See Also

GET ITEM TYPE FROM GKSM
READ ITEM FROM GKSM

READ ITEM FROM GKSM

Operating States

WSOP, WSAC, SGOP

Syntax

GRDITM (WKID, MIDRL, MLDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Open metafile input workstation identifier. More than one metafile input workstation can be open.
MIDRL	Integer	Read	Maximum length (in bytes) of the item data record, in the range 0 to *IDRL*. (*IDRL* is the length of the item data record returned by the function GET ITEM TYPE FROM GKSM.) If *MIDRL* = 0, DEC GKS skips the record. If *MIDRL* < *IDRL*, the excess is truncated. If *MIDRL* = *IDRL*, the full record is read.
MLDR	Integer	Read	Dimension of the item data record.
DATREC(MLDR)	Character*80	Write	Item's data record. This data record is passed to the INTERPRET ITEM function.

Description

The READ ITEM FROM GKSM function reads the data record of the current metafile item 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 GET ITEM TYPE FROM GKSM 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. To skip an item, specify the value 0 as the maximum record length.

After returning the data record to the application program, this function makes the next item in the metafile the current item.

See Also

GET ITEM TYPE FROM GKSM
INTERPRET ITEM

WRITE ITEM TO GKSM

Operating States

WSAC, SGOP

Syntax

GWITM (WKID, TYPE, IDRL, LDR, DATREC)

Argument	Data Type	Access	Description
WKID	Integer	Read	Active metafile output workstation identifier. More than one metafile output workstation can be activated.
TYPE	Integer	Read	Metafile item type.
IDRL	Integer	Read	Length (in bytes) of the item data record.
LDR	Integer	Read	Dimension of the data record array.
DATREC(LDR)	Character*80	Read	Item's data record.

Description

The WRITE ITEM TO GKSM function writes a user-defined data item record to a metafile.

You can 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 value greater than 100 to specify such a description for a GKSM file. The application program can treat any item type value greater than 100 as such a description. In a GKS3 file, values less than 0 indicate a user-defined item.

If you use a metafile structure that is different from the structure of a GKSM or GKS3, you may have to specify different item data record values to this function.

See Also

ACTIVATE WORKSTATION
OPEN WORKSTATION

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W