# DEC GKS

# C Binding Reference Manual, Part 1

Order Number: AA–MJ30C–TE

**June 1992**

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

**Revision/Update Information:** This revised manual supersedes the information in the *DEC GKS C Binding Reference Manual* (Order No. AA–MJ30B–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.



ZK5680

This manual is available on CDROM.

This document was prepared using VAX DOCUMENT, Version 2.1.

# Contents

## 3 ULTRIX Programming Considerations



## 4 Control Functions



## 5 Output Functions

## 6 Attribute Functions

## 7 Transformation Functions

## 8 Segment Functions



## 9 Input Functions

## 10 Metafile Functions



## Index

## Examples

## Figures

## Tables

# Preface

This manual contains complete descriptions for the C 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 C binding interface.
- Appendix C provides an example program written in C using the C binding.
- Appendix D lists the DEC GKS functions and the corresponding C 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 FORTRAN Binding Reference Manual*—for programmers who need specific syntax and argument descriptions for the FORTRAN 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

The C language binding of GKS follows the standards and conventions of VAX C™. For information on the C language, see the VAX C documentation set.

## 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 C binding is based on the ISO GKS DP 8651/4 (first draft) and is extended to three dimensions by Digital. Future versions of DEC GKS will conform to the ISO GKS and GKS–3D C binding standards when they become available.

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

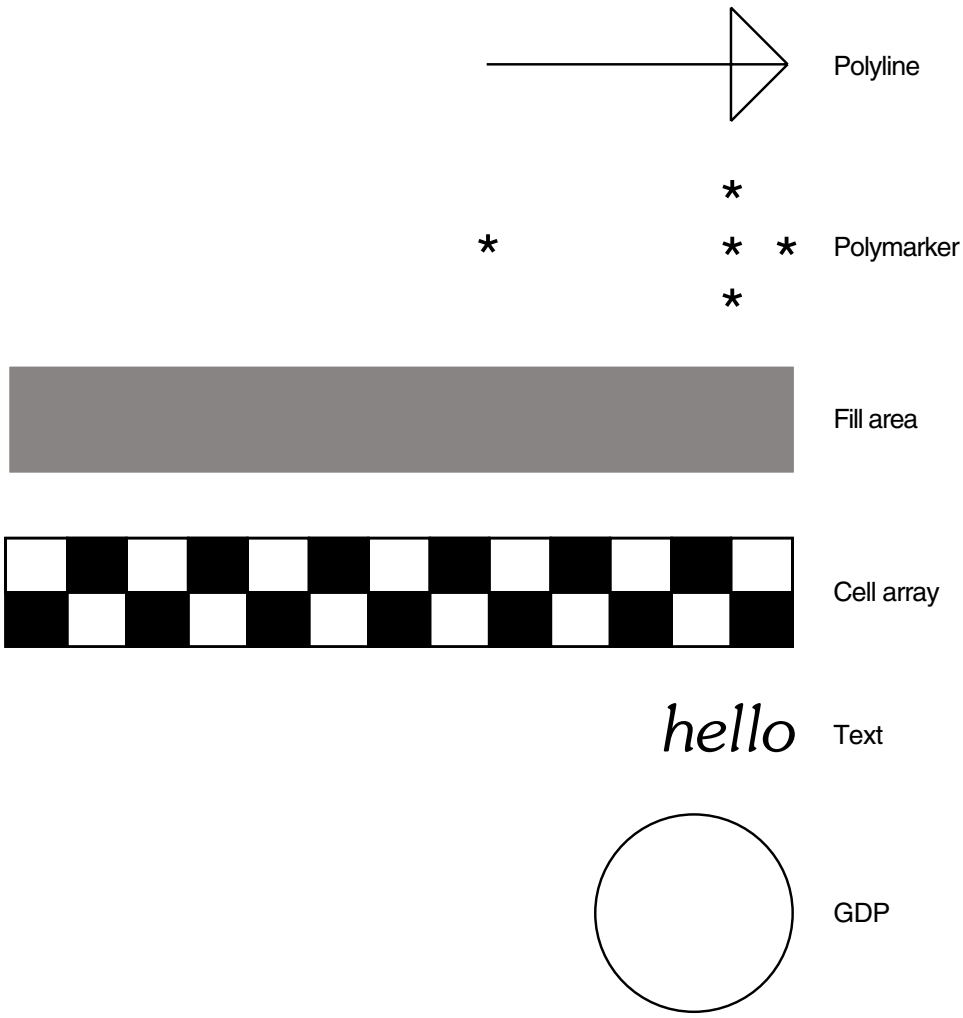
The control functions determine which DEC GKS functions you can call at a given point in your program. They also control the buffering of output and the regeneration of segments on the workstation surface.

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:

```
Gint     ws;     /* (I) 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. The access is marked as either I (input) or O (output).
- The argument-passing mechanism and form. If the argument is preceeded by an indirection symbol (*), the argument is passed by reference; otherwise it is passed by value. All structures and unions are passed by reference.

All the DEC GKS functions always return a longword condition status value. For a description of the longword status value, see Appendix A. For information concerning DEC GKS error handling, see Chapter 12.

### 1.3.4 Data Structures

The data structures section lists the DEC GKS data structures used by the function, from the top-level data structure down to the lower-level structures. The top-level structures are listed in order of appearance in the function call. Any enumerated types included in a data structure are listed in the Constants section; a marker specifies that the field is a constant.

### 1.3.5 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.

### 1.3.6 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.

### 1.3.7 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 of 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 C 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:

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

   gupdatews( ws_id, GPOSTPONE );
   gawaitevent( timeout, &event );
                .
                .
                .
```

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:

```
                .
                .
                .
default_conid = GWC_DEF;
default_wstype = GWS_DEF;

gopenws( ws_id, &default_conid, &default_wstype);
                .
                .
                .
```

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™ 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 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 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 workstation.

## 1.4 DEC GKS Compatibility

The DEC GKS Version 5.0 C binding is source-code compatible with the DEC GKS Version 4.2 C binding, with two exceptions. These exceptions are as follows:

- DEC GKS Version 5.0 functions return the errors as integers, based on the ISO standard and DEC (negative) error numbers.
- The DEC GKS Version 5.0 C binding now defines and requires function prototypes. Because of this change, DEC GKS may generate compilation warnings or errors if the application passes incorrect data types in the function calls.
- The DEC GKS Version 5.0 INQUIRE WORKSTATION CONNECTION IDENTIFIER AND TYPE function (ginqwsconntype) requires the Gwsct structure element *type* to be a pointer to an object of type Gwstype. The function returns the workstation type to this object. This is a change from earlier versions of DEC GKS that returned the element *type* erroneously in the Gwsct structure element *type*, instead of in the location pointed to by this element.

The DEC GKS Version 5.0 C binding is fully run-time compatible with the DEC GKS Version 4.2 C binding. Recompiling and relinking your application with DEC GKS Version 5.0 is recommended.

The DEC GKS Version 5.0 C binding is *not* source-code compatible with the DEC GKS–3D™ Version 1.2 C binding. However, a compatibility mode exists, which provides full source-code compatibility with DEC GKS–3D Version 1.2. See Section 1.6.1 for more information on the compatibility mode. The DEC GKS Version 5.0 C binding is fully run-time compatible with the DEC GKS–3D Version 1.2 C binding. Recompiling and relinking your application with DEC GKS Version 5.0 is recommended.

## 1.5 Porting DEC GKS Version 4.2 Applications

Check your application for DEC GKS Version 4.2 function return messages by searching for the strings "GKS$_SUCCESS" and "GKS$_ERROR". Replace these strings with the corresponding error constants defined in the C binding header file gks.h.

If your application uses the function INQUIRE WORKSTATION CONNECTION IDENTIFIER AND TYPE (ginqwsconntype), you must change your code to reflect a bug fix. The Version 5.0 function requires the Gwsct structure element *type* to be a pointer to an object of type Gwstype. The function returns the workstation type to this object. This is a change from earlier versions of DEC GKS that returned the element *type* erroneously in the Gwsct structure element *type*, instead of in the location pointed to by this element.

Recompile and relink your application with DEC GKS Version 5.0.

## 1.6 Porting DEC GKS–3D Version 1.2 Applications

The following sections describe the two options for porting DEC GKS–3D Version 1.2 applications:

- Compatibility mode
- Manual porting

### 1.6.1 Using Compatibility Mode

You can enable the DEC GKS–3D compatibility mode using one of the following header files:

- GKS3D$CBND.H (VMS)
- GKS3Dcbnd.h (ULTRIX)
- gks.h header file with the GKS3D compile-time constant

The compile-time constant can be defined as part of the compile command (see your compiler's manual), or by adding the statement `#define GKS3D` before the `#include <gks.h>` statement.

Using compatibility mode, no changes are required to the source code if the following restrictions are true:

- Your DEC GKS application uses only the C binding enumeration constants, not hardcoded integer values, as defined in the DEC GKS C binding header file.
- Your DEC GKS application does not call any of the INQUIRE DEFAULT . . . DEVICE DATA functions.

If you use hardcoded integer values, you must update the values for the following enumerations accordingly:

```
typedef enum {          /* CLIPPING INDICATOR */
    GCLIP,
    GNOCLIP
} Gclip;

typedef enum {          /* CHOICE STATUS */
    GC_OK,
    GC_NOCHOICE,
    GC_NONE
} Gcstat;

typedef enum {          /* ECHO SWITCH */
    GECHO,
    GNOECHO
} Gesw;

typedef enum {          /* REQUEST STATUS */
    GOK,
    GNONE
} Gistat;

typedef enum {          /* PICK STATUS */
    GP_OK,
    GP_NOPICK,
    GP_NONE
} Gpstat;
```

If your program calls any of the INQUIRE DEFAULT . . . DEVICE DATA functions, you must update these calls according to the function definitions in this manual.

## 1.6.2 Using Manual Porting

If you do not use compatibility mode, you must make the changes described in the following sections to port your code to DEC GKS Version 5.0. After you complete the adjustments, recompile and relink the application with DEC GKS Version 5.0.

### 1.6.2.1 Changes to Choice Input Devices

You must make the following modifications to the application source code for it to compile properly:

- Rename Gchoicepetneg0001 to Gchoicepet_0001.
- In the data structures Gchoicepet0001, Gchoicepet0003, Gchoicepet0004, and Gchoicepet0005, replace the *data* field with the *title_string* field. This field addresses the title string. In the data structures Gchoicepet0003 and Gchoicepet0004, remove the *lengths* field. The data structures are as follows:

```
typedef struct {
    Gint      number;         /* number of choice strings */
    Gint      *lengths;       /* lengths of choice strings */
    Gchar     **strings;      /* array of strings */
    Gchar     *title_string;  /* the title string */
} Gchoicepet0001;

typedef struct {
    Gint    number;        /* number of choice strings */
    Gchar   **strings;     /* array of strings */
    Gchar   *title_string; /* the title string */
} Gchoicepet0003;

typedef Gchoicepet0003 Gchoicepet0004;

typedef struct {
    Gint    seg;           /* segment name */
    Gint    number;        /* number of alternatives */
    Gint    *pickids;      /* array of pick identifiers */
    Gchar   *title_string; /* the title string */
} Gchoicepet0005;
```

- The Gchoicepet0002 data structure is as follows:

```
typedef struct {
    Gint    number;        /* number of alternatives */
    Gprflag *enable;       /* array of prompts */
    Gchar   *title_string; /* title string */
} Gchoicepet0002;
```

- In the Gchoicerec data structure, replace the field *choicepetneg1_datarec* of type Gchoicepetneg0001 with the *choicepet_1_datarec* field of type Gchoicepet_0001. The data structure is as follows:

```
typedef union {         /* CHOICE DATA RECORD */
    Gchoicepet_0001      choicepet_1_datarec;
    Gchoicepet0001       choicepet1_datarec;
    Gchoicepet0002       choicepet2_datarec;
    Gchoicepet0003       choicepet3_datarec;
    Gchoicepet0004       choicepet4_datarec;
    Gchoicepet0005       choicepet5_datarec;
} Gchoicerec;
```

#### 1.6.2.2 Changes to Locator Input Devices

You must make the following additional modifications to the application source code for it to compile properly:

- Rename the Glocpetneg*nn* data structures (where *nn* represents the numbers 01 to 12) to Glocpet_00*nn*, respectively.
- In data structure Glocpet0006, replace the *data* field with the *title_string* field.
- In the data structure Glocpetneg0001, replace the *size_x* and *size_y* fields with the *box_x* and *box_y* fields, respectively. The data structure is as follows:

```
typedef struct {
    Gfloat   box_x;          /* size of the box in x */
    Gfloat   box_y;          /* size of the box in y */
    Gchar    *data;          /* device/implementation dependent data */
} Glocpetneg0001;
```

- In the Glocrec data structure, delete the field *locpetneg13_datarec*. Replace each of the twelve fields *locpetnegnn_datarec* (where *nn* represents the numbers 01 to 12) of type Glocpetneg00*nn* with the corresponding fields *locpet_nn_datarec* of type *Glocpet_000nn*, respectively. The data structure is as follows:

```
typedef union {          /* LOCATOR DATA RECORD */
    Glocpet_0001         locpet_1_datarec;
    Glocpet_0002         locpet_2_datarec;
    Glocpet_0003         locpet_3_datarec;
    Glocpet_0004         locpet_4_datarec;
    Glocpet_0005         locpet_5_datarec;
    Glocpet_0006         locpet_6_datarec;
    Glocpet_0007         locpet_7_datarec;
    Glocpet_0008         locpet_8_datarec;
    Glocpet_0009         locpet_9_datarec;
    Glocpet_0010         locpet_10_datarec;
    Glocpet_0011         locpet_11_datarec;
    Glocpet_0012         locpet_12_datarec;
    Glocpet0001          locpet1_datarec;
    Glocpet0002          locpet2_datarec;
    Glocpet0003          locpet3_datarec;
    Glocpet0004          locpet4_datarec;
    Glocpet0005          locpet5_datarec;
    Glocpet0006          locpet6_datarec;
} Glocrec;
```

#### 1.6.2.3 Changes to String Input Devices

You must make the following additional modifications to the application source code for it to compile properly:

- In the data structure Gstringpet0001, replace the *data* field with the *title_string* field. This field addresses the title string. The data structure is as follows:

```
typedef struct {
    Gint     bufsiz;         /* buffer size */
    Gint     position;       /* initial cursor position */
    Gchar    *title_string;  /* the title string */
} Gstringpet0001;
```

#### 1.6.2.4 Changes to Valuator Input Devices

You must make the following additional modifications to the application source code for it to compile properly:

- Rename the Gvalpetneg000*n* structures (where *n* is 1, 2, or 3) to Gvalpet_000*n*, respectively.
- In Gvalpet0001, replace the *data* field with the *title_string* field. This field addresses the title string. The data structure is as follows:

```
typedef struct {
    Gfloat  low;            /* low range limit */
    Gfloat  high;           /* high range limit */
    Gchar   *title_string;  /* the title string */
} Gvalpet0001;
```

- In the Gvalrec data structure, replace each of the three fields *valpetnegn_datarec* (where *n* represents a number between 1 and 3) of type Gvalpetneg000*n* with corresponding fields *valpet_n_datarec* of type Gvalpet_000*n*, respectively. The data structure is as follows:

```
typedef union {      /* VALUATOR DATA RECORD */
    Gvalpet_0001       valpet_1_datarec;
    Gvalpet_0002       valpet_2_datarec;
    Gvalpet_0003       valpet_3_datarec;
    Gvalpet0001        valpet1_datarec;
    Gvalpet0002        valpet2_datarec;
    Gvalpet0003        valpet3_datarec;
} Gvalrec;
```

#### 1.6.2.5 Changes to the Gcobundl Data Structure

Replace the three fields *comp1*, *comp2*, and *comp3*, with *red*, *green*, and *blue*, respectively. The data structure is as follows:

```
typedef struct {
    Gfloat   red;    /* red intensity */
    Gfloat   green;  /* green intensity */
    Gfloat   blue;   /* blue intensity */
} Gcobundl;
```

#### 1.6.2.6 Changes to Enumeration Types

There are several differences between the enumeration types in DEC GKS–3D Version 1.2 and DEC GKS Version 5.0. These differences are as follows:

| Enumeration | DEC GKS Version 5.0 | DEC GKS–3D Version 1.2 |
|---|---|---|
| Gclip | {GCLIP, GNOCLIP} | {GNOCLIP, GCLIP} |
| Gcstat | {GC_OK, GC_NOCHOICE, GC_NONE} | {GC_NONE, GC_OK, GC_ NOCHOICE} |
| Gesw | {GECHO, GNOECHO} | {GNOECHO, GECHO} |
| Gistat | {GOK, GNONE} | {GNONE, GOK} |
| Gpstat | {GP_OK, GP_NOPICK, GP_NONE} | {GP_NONE, GP_OK, GP_NOPICK} |

#### 1.6.2.7 Changes to the Input Inquiry Functions

The DEC GKS Version 5.0 arguments *bufsize* and *data_size* are different from DEC GKS–3D for the following functions:

- INQUIRE DEFAULT CHOICE DEVICE DATA
- INQUIRE DEFAULT CHOICE DEVICE DATA 3
- INQUIRE DEFAULT LOCATOR DEVICE DATA
- INQUIRE DEFAULT LOCATOR DEVICE DATA 3
- INQUIRE DEFAULT PICK DEVICE DATA
- INQUIRE DEFAULT PICK DEVICE DATA 3
- INQUIRE DEFAULT STRING DEVICE DATA
- INQUIRE DEFAULT STRING DEVICE DATA 3
- INQUIRE DEFAULT STROKE DEVICE DATA
- INQUIRE DEFAULT STROKE DEVICE DATA 3
- INQUIRE DEFAULT VALUATOR DEVICE DATA
- INQUIRE DEFAULT VALUATOR DEVICE DATA 3

Check your program for these function calls and update them according to the definitions in the DEC GKS Version 5.0 C binding manual.

**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$HELP:DECGKS_CBIND_OP_SPEC.PS
SYS$HELP:DECGKS_CBIND_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.

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

```
# include <gks.h>
```

In the previous statement, the angle brackets (< >) show the files containing DEC GKS constants are contained in the system library.

The language definition file located in SYS$LIBRARY is gks.h.

The file includes comments that describe the exact method for using the definition file.

## 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 either of the following DCL commands:

```
$ LINK program, SYS$LIBRARY:GKSCBND.OLB, - RETURN
SYS$INPUT:/OPTIONS  RETURN
SYS$LIBRARY:VAXCRTL/SHARE  CTRL/Z

$ LINK program, SYS$LIBRARY:GKS_CBND/LIB  -  RETURN
SYS$LIBRARY:VAXCRTL/LIB  CTRL/Z
```

## 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 in any of the following ways:

- By logical name: Pass a logical name specifying the connection ID.
- By file or device name: Pass the connection ID as a string.
- By default: Pass either the value 0 or a null string. DEC GKS then attempts to translate the logical name GKS$CONID. If no translation exists, GKS uses GKS_DEFAULT.OUTPUT, which specifies a file in the current directory as the connected device. The user can define the VMS logical name GKS$CONID.

### 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$LIBRARY:GKS.H
- Use the default workstation type by passing a value of 0. DEC GKS attempts to translate the VMS logical name GKS$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.h.

The function call can include a check of the returned status. The value NO_ERROR (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.

### 2.7.2 Error Files

Error messages are normally written to an error logging file. The application can specify the error logging file by passing the file name to the OPEN GKS function in either of the following ways:

- Explicitly, by specifying the predefined file pointer: The file is specified by a standard C file pointer that is returned by the fopen( ) system call. The

application must open and close the error file, respectively, before opening and closing DEC GKS.

- By default, by passing a null file pointer.

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 ULTRIX™ C (cc) 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_cbind_op_spec.ps
/usr/lib/GKS/doc/decgks_cbind_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 C binding is /usr/include/GKS/gks.h.

The file includes comments that describe how to use the language definition file.

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

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

## 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 the Program on ULTRIX Systems with RISC Processors

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

```
# cc -I/usr/include/GKS -o program program.c [gksconfig.c] \ RETURN
-lGKScbnd -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 -lc -lX11 -llmf -lm RETURN
```

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

```
# cc -I/usr/include/GKS -o program program.c gks_decw_config.c \ RETURN
-lGKScbnd -lGKS -lddif -ldwt -lcursesX -lc -lX11 -llmf -lm RETURN
```

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

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 device handler. It is located in /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:

- By environment variable: Pass an environment variable specifying the connection ID.
- By file or device name: Pass the connection ID as a string.

- By default: Pass either the value 0 or a null string. DEC GKS then attempts to translate the environment variable GKSconid. If no translation exists, GKS uses gks_default.output, which specifies a file in the current directory as the connected device. To specify an environment option, the user can do either of the following:
  - Define the environment variable GKSconid.
  - Use the user defaults file ~/.GKSdefaults, or the system defaults file /usr/lib/GKS/.GKSdefaults.

### 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 language file /usr/include/GKS/GKScbnd.h.
- 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). The user can do either of the following:
  - Define the environment variable GKSwstype.
  - Use the user defaults file ~/.GKSdefaults, or the system defaults file /usr/lib/GKS/.GKSdefaults.

## 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.h.

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.

### 3.8.2 Error Files

Error messages are normally written to an error logging file. The application can specify the error logging file by passing the file name to the OPEN GKS function in either of the following ways:

- Explicitly, by specifying the predefined file pointer: The file is specified by a standard C file pointer that is returned by the fopen( ) system call. The application must open and close the error file, respectively, before opening and after closing DEC GKS.
- By default, by passing a null file pointer.

DEC GKS opens the default error file. To do this, it first attempts to translate GKSerrfile. If no translation exists, DEC GKS uses stderr as the file pointer. The user can do either of the following:

- Define the environment variable GKSerrfile.
- Use the user defaults file ~/.GKSdefaults, or the system defaults file /usr/lib/GKS/.GKSdefaults.

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 allows 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 directory /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 |
|---|---|
| GOUTPUT | A workstation of the category GOUTPUT 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 GOUTPUT 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 GPOSTPONE. 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 (GALLOWED) or suppressing them (GSUPPRESSED).

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 GPERFORM, 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 GNOTEMPTY when the surface is actually empty. In such situations, when the entry contains GNOTEMPTY, 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 GPERFORM 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 GPERFORM 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

gactivatews (

```
  Gint     ws    /* (I) 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

gclearws (

```
  Gint       ws,           /* (I) Workstation identifier */
  Gclrflag   clearflag     /* (I) Clear screen flag (constant) */
```

)

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gclrflag | GCONDITIONALLY | Clear if the surface is not empty. |
| | GALWAYS | 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

gclosegks ( )

### 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

gclosews (

```
  Gint   ws    /* (I) 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

gdeactivatews (

```
  Gint   ws    /* (I) 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

gescape (

```
  Gint      function,      /* (I) Escape function identifier (constant).*/
  Gescin    *indata,       /* (I) Pointer to input data record. */
  Gint      bufsize,       /* (I) Size of input data record (bytes).
                                 Must be the exact size of the input
                                 data record. */
  Gescout   *outdata,      /* (O) Pointer to output data record. */
  Gint      *escout_size   /* (O) Size of output record (bytes). */
```

)

## Data Structures

```
typedef union {                         /* ESCAPE INPUT STRUCTURE */
  Guesc_idatarec   esc_idatarec;
} Gescin;

    typedef struct {                    /* escape-dependent data record */
      Gint      number_integer;         /* number of integers */
      Gint      number_float;           /* number of floats */
      Gint      number_strings;         /* number of strings */
      Gint      *list_integers;         /* list of integers */
      Gfloat    *list_floats;           /* list of floats */
      Gint      *list_string_lengths;   /* list of string lengths */
      Gchar     **list_strings;         /* list of strings */
    } Guesc_idatarec;

typedef union {                         /* ESCAPE OUTPUT STRUCTURE */
  Guesc_odatarec     esc_odatarec;
} Gescout;

    typedef struct {                    /* escape-dependent data record */
      Gint      number_integer;         /* number of integers */
      Gint      number_float;           /* number of floats */
      Gint      number_strings;         /* number of strings */
      Gint      *list_integers;         /* list of integers */
      Gfloat    *list_floats;           /* list of floats */
      Gint      *list_string_lengths;   /* list of string lengths */
      Gchar     **list_strings;         /* list of strings */
    } Guesc_odatarec;
```

## Constants

| Escape Identifier | Description |
|---|---|
| ESC_ESP | Set Display Speed |
| ESC_EP | Generate Hardcopy of Workstation Surface |
| ESC_EB | Beep |
| ESC_EPOPW | Pop Workstation |
| ESC_EPSHW | Push Workstation |
| ESC_ESEHM | Set Error Handling Mode |
| ESC_ESVE | Set Viewport Event |
| ESC_EAWC | Associated Workstation Type and Connection ID |
| ESC_ESCL | Software Clipping |
| ESC_ESWM | Set Writing Mode |
| ESC_ESLC | Set Line Cap Style |
| ESC_ESLJ | Set Line Join Style |
| ESC_ESEC | Set Edge Control Flag |
| ESC_ESET | Set Edge Type |
| ESC_ESEW | Set Edge Width Scale Factor |
| ESC_ESECI | Set Edge Color Index |
| ESC_ESEI | Set Edge Index |
| ESC_ESEA | Set Edge Aspect Source Flag |
| ESC_EBTB | Begin Transformation Block |
| ESC_EETB | End Transformation Block |
| ESC_ESSHM | Set Segment Highlighting Method |
| ESC_ESHM | Set Highlighting Method |
| ESC_BTB3 | Begin Transformation Block 3 |
| ESC_ESER | Set Edge Representation |
| ESC_ESWT | Set Window Title |
| ESC_ESRS | Set Reset String |
| ESC_ESCS | Set Cancel String |
| ESC_ESES | Set Enter String |
| ESC_ESIB | Set Icon Bit Maps |
| ESC_EIWM | Inquire Current Writing Mode |
| ESC_EILC | Inquire Current Line Cap Style |
| ESC_EILJ | Inquire Current Line Join Style |
| ESC_EIEA | Inquire Current Edge Attributes |
| ESC_EIVD | Inquire Viewport Data |
| ESC_EIS | Inquire Current Display Speed |
| ESC_EILEI | Inquire List of Edge Indexes |
| ESC_EISE | Inquire Segment Extent |
| ESC_EIWID | Inquire Window Identifiers |
| ESC_EISHM | Inquire Segment Highlighting Method |
| ESC_EIHM | Inquire Highlighting Method |
| ESC_EIPBI | Inquire Pasteboard Identifier |
| ESC_EIMBI | Inquire Menu Bar Identifier |
| ESC_EISHI | Inquire Shell Identifier |
| ESC_EILE | Inquire List of Available Escapes |
| ESC_EIDS | Inquire Default Display Speed |
| ESC_EILCJ | Inquire Line Cap and Join Facilities |
| ESC_EIEF | Inquire Edge Facilities |
| ESC_EIPER | Inquire Predefined Edge Representation |
| ESC_EIMEB | Inquire Maximum Number of Edge Bundles |
| ESC_EILH | Inquire List of Highlighting Methods |
| ESC_IER | Inquire Edge Representation |
| ESC_EMNW | Evaluate NDC Mapping of a WC Point |
| ESC_EMDN | Evaluate DC Mapping of an NDC Point |
| ESC_EMWN | Evaluate WC Mapping of an NDC Point |
| ESC_EMND | Evaluate NDC Mapping of a DC Point |
| ESC_EIGEX | Inquire Extent of a GDP |
| ESC_ESDB | Set Double Buffering |
| ESC_ESBPM | Set Background Pixmap |

```
ESC_EIDBM           Inquire Double Buffer Pixmap
ESC_EIBGM           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 this function call to produce a hardcopy dump of a VT125 or VT240 terminal screen, or to set the LVP16™ plotter pen speed.

When calling the control function ESCAPE, you may need to pass a data record. There are two types of escape data records, input and output. The C escape input data record is as follows:

```
typedef struct {                    /* escape-dependent data record */
  Gint      number_integer;         /* number of integers */
  Gint      number_float;           /* number of floats */
  Gint      number_strings;         /* number of strings */
  Gint      *list_integers;         /* list of integers */
  Gfloat    *list_floats;           /* list of floats */
  Gint      *list_string_lengths;   /* list of string lengths */
  Gchar     **list_strings;         /* list of strings */
} Guesc_idatarec;
```

The C escape output data record is as follows:

```
typedef struct {                    /* escape-dependent data record */
  Gint      number_integer;         /* number of integers */
  Gint      number_float;           /* number of floats */
  Gint      number_strings;         /* number of strings */
  Gint      *list_integers;         /* list of integers */
  Gfloat    *list_floats;           /* list of floats */
  Gint      *list_string_lengths;   /* list of string lengths */
  Gchar     **list_strings;         /* list of strings */
} Guesc_odatarec;
```

See the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS* for the input and output values required by each escape. Both the input and output data records must be completed as described.

The argument *bufsize* is the sizeof (Guesc_idatarec). If an output data record is necessary, you can either use the output data size number (as described in the *Device Specifics Reference Manual for DEC GKS and DEC PHIGS*), or you can determine the size using the ESCAPE function.

To use the ESCAPE function to determine the size of the entire output data record buffer, pass the value 0 as the argument *escout_size* (size of the output data record). DEC GKS checks for errors, returns the size of the output data record in *escout_size*, but does *not* perform the escape. Call the function a second time with the correct output data record size to have DEC GKS perform the escape.

## See Also

Example 4–2 for a program example using the ESCAPE function

# MESSAGE

## Operating States

WSOP, WSAC, SGOP

## Syntax

gmessage (

```
  Gint      ws,          /* (I) Workstation identifier */
  Gchar     *message     /* (I) Pointer to the message text */
```

)

## 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

gopengks (

```
  Gfile    *error_fp,     /* (I) Error file pointer.  This is either a
                                 logical name or a physical name of a
                                 device or file that points to the error
                                 log file.  The default value is 0. */
  Glong     memory        /* (I) Memory allocation.  This is a dummy
                                 argument required by the 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

gopenws (

```
  Gint        ws,       /* (I) Workstation identifer.  This must be a
                               positive integer value. */
  Gconn       *conid,   /* (I) Connection identifier.  This can be a
                               logical name, a DEC GKS constant value,
                               or a file specification. */
  Gwstype     *wstype   /* (I) 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.

## See Also

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

## REDRAW ALL SEGMENTS ON WORKSTATION

### Operating States

WSOP, WSAC, SGOP

### Syntax

gredrawsegws (

```
  Gint    ws     /* (I) 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

gsetdeferst (

```
  Gint        ws,         /* (I) Workstation identifier */
  Gdefmode    defmode,    /* (I) Maximum deferral mode allowed (constant) */
  Girgmode    irgmode     /* (I) Implicit regeneration mode (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gdefmode | 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. |
| Girgmode | GSUPPRESSED | Image regeneration is suppressed. |
| | GALLOWED | 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

gupdatews (

```
  Gint       ws,           /* (I) Workstation identifier */
  Gregen     regenflag     /* (I) Update regeneration mode (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gregen | GPERFORM | Perform regeneration of image. |
| | GPOSTPONE | Postpone 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 GPERFORM 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 GPOSTPONE 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**

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

# include <stdio.h>
# include <gks.h>             /* C binding definitions file */

main ()
{
   Gconn     default_conid;
   Gwstype   default_wstype;
   Gevent    event;
   Gfloat    larger     =  0.03;
   Gpoint    position;
   Gchar    *text_name  = "CLEAR WORKSTATION should erase this";
   Gfloat    timeout    =  10.0;
   Gint      ws_id      =  1;

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

    gopengks (0, 0);

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

    default_conid  = GWC_DEF;
    default_wstype = GWS_DEF;

    gopenws (ws_id, &default_conid, &default_wstype);

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

    gactivatews (ws_id);

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

    position.x = 0.1;
    position.y = 0.5;
```

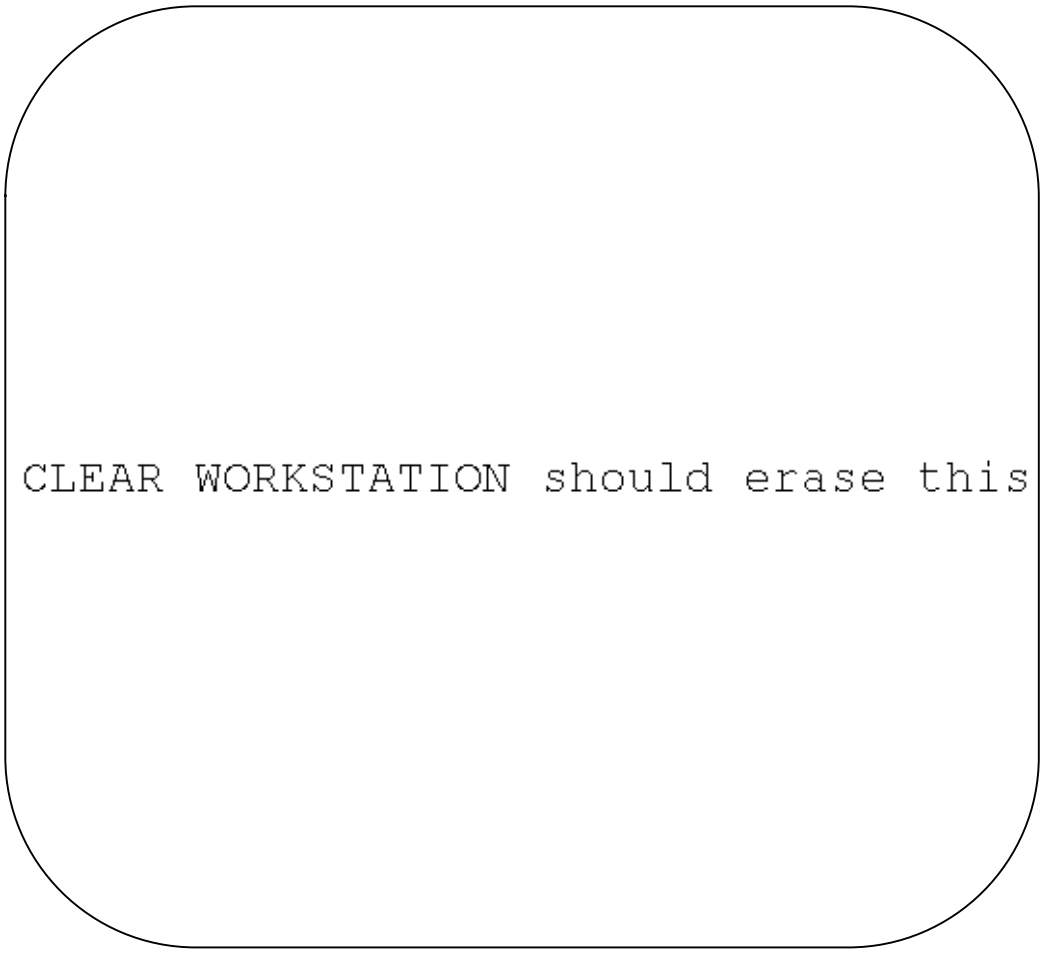
(continued on next page)

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

```
    gsetcharheight (larger);
    gtext (&position , text_name);

/*  Release the deferred output.  Wait 10 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout, &event);

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

    gclearws (ws_id, GCONDITIONALLY);

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

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();
}
```

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**

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

# include <stdio.h>
# include <gks.h>           /* GKS C binding definition file */

# define FALSE         0
# define MAX_BUFFER   80
# define MAX_INTS    100
# define TRUE          1
```

(continued on next page)

**Example 4–2 (Cont.) Supported Escapes Program**

```
/*
 *  This routine inquires the list of supported escapes for the specified
 *  workstation type.
 */

void inquire_escapes (wstype, escape_list)

Gwstype   wstype;
Gint      escape_list[MAX_INTS];

{
    Gint    in_data_size;
    Gescin  in_esc_data;
    Gint    out_data_size;
    Gescout out_esc_data;

    in_data_size = sizeof(Gescin);
    in_esc_data.esc_idatarec.number_integer = 1;
    in_esc_data.esc_idatarec.number_float = 0;
    in_esc_data.esc_idatarec.number_strings = 0;
    in_esc_data.esc_idatarec.list_integers = &wstype;
    in_esc_data.esc_idatarec.list_floats = NULL;
    in_esc_data.esc_idatarec.list_string_lengths = NULL;
    in_esc_data.esc_idatarec.list_strings = NULL;

    out_data_size = sizeof(Gescout);
    out_esc_data.esc_odatarec.number_integer = MAX_INTS; /* maximum array
                                                            dimension */
    out_esc_data.esc_odatarec.number_float = 0;
    out_esc_data.esc_odatarec.number_strings = 0;
    out_esc_data.esc_odatarec.list_integers = escape_list;
    out_esc_data.esc_odatarec.list_floats = NULL;
    out_esc_data.esc_odatarec.list_string_lengths = NULL;
    out_esc_data.esc_odatarec.list_strings = NULL;

    gescape (ESC_EILE, &in_esc_data, in_data_size,
       &out_esc_data, &out_data_size);

} /* End inquire_escapes */

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

int esc_support (esc_id, escape_list)

Gint   esc_id;
Gint   escape_list[MAX_INTS];

{
    int   escape_supported;
    int   n;

    escape_supported = FALSE;
     for (n = 3; n < 3+escape_list[2]; n++)
       if (esc_id == escape_list[n])
           {
           escape_supported = TRUE;
           break;
           }

    return (escape_supported);
```

(continued on next page)

**Example 4–2 (Cont.) Supported Escapes Program**

```
} /* End esc_support */

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

void set_double_buffer (ws_id, escape_list, double_buffer_flag)

Gint   double_buffer_flag;
Gint   escape_list[MAX_INTS];
Gint   ws_id;

{
    Gint    in_data_size;
    Gescin  in_esc_data;
    Gint    int_array[2];
    Gint    out_data_size;
    Gescout out_esc_data;

/*  Determine if the escape is supported by workstation. */

    if (!esc_support(ESC_ESDB, escape_list))
         return;

/*  Turn double buffering ON or OFF as requested. */

    in_data_size = sizeof(Gescin);
    in_esc_data.esc_idatarec.number_integer = 2;
    in_esc_data.esc_idatarec.number_float = 0;
    in_esc_data.esc_idatarec.number_strings = 0;
    in_esc_data.esc_idatarec.list_integers = int_array;
    in_esc_data.esc_idatarec.list_floats = NULL;
    in_esc_data.esc_idatarec.list_string_lengths = NULL;
    in_esc_data.esc_idatarec.list_strings = NULL;
    int_array[0] = ws_id;
    int_array[1] = double_buffer_flag;


    out_data_size = sizeof(Gescout);
    out_esc_data.esc_odatarec.number_integer = 0;
    out_esc_data.esc_odatarec.number_float = 0;
    out_esc_data.esc_odatarec.number_strings = 0;
    out_esc_data.esc_odatarec.list_integers = NULL;
    out_esc_data.esc_odatarec.list_floats = NULL;
    out_esc_data.esc_odatarec.list_string_lengths = NULL;
    out_esc_data.esc_odatarec.list_strings = NULL;

    gescape (ESC_ESDB, &in_esc_data, in_data_size,
       &out_esc_data, &out_data_size);

} /* End set_double_buffer */

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

void inq_window_ids (ws_id, escape_list, x_display_id, x_window_id)

Gint   escape_list[MAX_INTS];
Gint   ws_id;
Gint   *x_display_id;
Gint   *x_window_id;
```

(continued on next page)

**Example 4–2 (Cont.) Supported Escapes Program**

```
{
    Gint    in_data_size;
    Gescin  in_esc_data;
    Gint    int_array[2];
    Gint    out_data_size;
    Gescout out_esc_data;

/*  Determine if the escape is supported by the workstation. */

    if (!esc_support(ESC_EIWID, escape_list))
        return;

/*  Inquire the X display identifier and the X window identifier. */

    in_data_size = sizeof(Gescin);
    in_esc_data.esc_idatarec.number_integer = 1;
    in_esc_data.esc_idatarec.number_float = 0;
    in_esc_data.esc_idatarec.number_strings = 0;
    in_esc_data.esc_idatarec.list_integers = &ws_id;
    in_esc_data.esc_idatarec.list_floats = NULL;
    in_esc_data.esc_idatarec.list_string_lengths = NULL;
    in_esc_data.esc_idatarec.list_strings = NULL;

    out_data_size = sizeof(Gescout);
    out_esc_data.esc_odatarec.number_integer = 2;
    out_esc_data.esc_odatarec.number_float = 0;
    out_esc_data.esc_odatarec.number_strings = 0;
    out_esc_data.esc_odatarec.list_integers = int_array;
    out_esc_data.esc_odatarec.list_floats = NULL;
    out_esc_data.esc_odatarec.list_string_lengths = NULL;
    out_esc_data.esc_odatarec.list_strings = NULL;

    gescape (ESC_EIWID, &in_esc_data, in_data_size,
       &out_esc_data, &out_data_size);

    *x_display_id = int_array[0];
    *x_window_id  = int_array[1];

    printf ("\n Escape: Inquire Window Identifiers\n");
    printf ("   X Display ID:      %x\n", *x_display_id);
    printf ("   X Window ID:       %x\n", *x_window_id);

} /* End inq_window_ids */

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

void set_window_title (ws_id, escape_list, new_title)

Gint   escape_list[MAX_INTS];
Gchar  *new_title;
Gint   ws_id;

{
    Gint    in_data_size;
    Gescin  in_esc_data;
    Gint    new_title_size;
    Gint    out_data_size;
    Gescout out_esc_data;
```

(continued on next page)

**Example 4–2 (Cont.) Supported Escapes Program**

```
/*  Determine if the escape is supported by the workstation. */

    if (!esc_support(ESC_ESWT, escape_list))
        return;

/*  Change the window title. */

    in_data_size = sizeof(Gescin);
    in_esc_data.esc_idatarec.number_integer = 1;
    in_esc_data.esc_idatarec.number_float = 0;
    in_esc_data.esc_idatarec.number_strings = 1;
    in_esc_data.esc_idatarec.list_integers = &ws_id;
    in_esc_data.esc_idatarec.list_floats = 0;
    in_esc_data.esc_idatarec.list_string_lengths = &new_title_size;
    in_esc_data.esc_idatarec.list_strings = &new_title;
    new_title_size = strlen(new_title);


    out_data_size = sizeof(Gescout);
    out_esc_data.esc_odatarec.number_integer = 0;
    out_esc_data.esc_odatarec.number_float = 0;
    out_esc_data.esc_odatarec.number_strings = 0;
    out_esc_data.esc_odatarec.list_integers = NULL;
    out_esc_data.esc_odatarec.list_floats = NULL;
    out_esc_data.esc_odatarec.list_string_lengths = NULL;
    out_esc_data.esc_odatarec.list_strings = NULL;

    gescape (ESC_ESWT, &in_esc_data, in_data_size,
       &out_esc_data, &out_data_size);

} /* End set_window_title */

main ( )
{

   Gint      double_buffer_flag;
   Gint      error_ind;
   Gint      escape_list [MAX_INTS];
   Gchar     inq_buffer [MAX_BUFFER];
   Gint      n;
   Gwsct     ws_con_type;
   Gint      ws_con_type_size;
   Gint      ws_id = 1;
   Gwstype   wstype;
   Gint      x_display_id;
   Gint      x_window_id;

/*  Open GKS and the default workstation. */

    gopengks (stderr, 0);
    gopenws (ws_id, 0, 0);

/*  Inquire the workstation type. */

    ws_con_type.conn = inq_buffer;
    ws_con_type.type = &wstype;

    ginqwsconntype (ws_id, MAX_BUFFER, &ws_con_type_size,
       &ws_con_type, &error_ind);

/*  Inquire the list of supported escapes. */
```

(continued on next page)

**Example 4–2 (Cont.) Supported Escapes Program**

```
    inquire_escapes (wstype, escape_list);
      if (escape_list[0] != 0)
         {
         gemergencyclosegks ( );
         printf ("Error inquiring list of supported escapes %d\n",
                 escape_list[0]);
         exit ( );
         }

/*  Set the window title. */

    set_window_title (ws_id, escape_list, "DEC GKS puts life in perspective");

/*  Turn on double buffering. */

    double_buffer_flag = 1;

    set_double_buffer (ws_id, escape_list, double_buffer_flag);

/*  Inquire the X display identifier and X window identifier. */

    inq_window_ids (ws_id, escape_list, &x_display_id, &x_window_id);

/*  Close the workstation and GKS. */

    gclosews (ws_id);
    gclosegks ( );

/*  Print the workstation type and its supported escapes. */

    printf ("\n Workstation type %d supports %d escapes (%d returned):\n",
             wstype, escape_list[1], escape_list[2]);
      for (n = 3; n < escape_list[2] + 3; n++)
         printf ("    %d\n", escape_list[n]);

} /* End main */
```

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
```

**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

gcellarray (

```
  Grect   *rectangle,     /* (I) Upper left/lower right corners */
  Gidim   *dimensions,    /* (I) Color index array dimensions */
  Gint    *colour         /* (I) Color indexes in row order form */
```

)

## Data Structures

```
typedef struct {      /* COORDINATE RECTANGLE */
  Gpoint    ul;       /* upper left corner */
  Gpoint    lr;       /* lower right corner */
} Grect;

    typedef struct {  /* COORDINATE POINT */
      Gfloat    x;    /* X coordinate */
      Gfloat    y;    /* Y coordinate */
    } Gpoint;

typedef struct {      /* DIMENSION IN INTEGER VALUES */
  Gint      x_dim;    /* X dimension */
  Gint      y_dim;    /* Y dimension */
} Gidim;
```

## Description

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

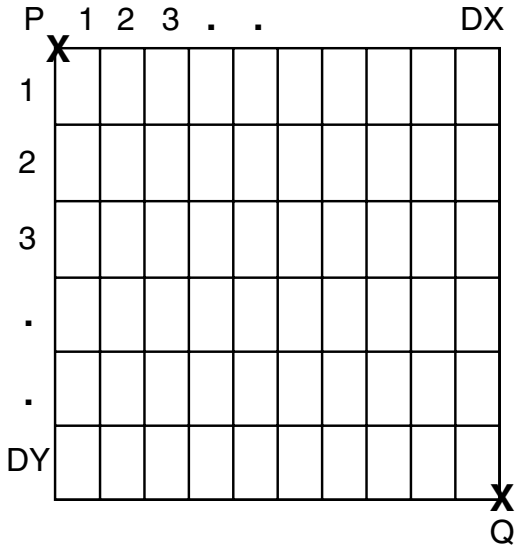
You pass a two-dimensional array containing color index values as one argument to this function. DEC GKS maps the color index values to corresponding cells within a rectangular area of the workstation surface. In addition to the color index array, you specify the number of array columns and rows to be 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

gcellarray3 (

```
  Grect3    *parallelogram, /* (I) Three vertices of a parallelogram */
  Gidim     *dimensions,    /* (I) Color index array dimensions */
  Gint      *colour         /* (I) Color indexes in row order form */
```

)

## Data Structures

```
typedef struct {      /* WORLD COORDINATE PARALLELOGRAM */
  Gpoint3   llf;      /* lower left front corner */
  Gpoint3   urf;      /* upper right front corner */
  Gpoint3   urb;      /* upper right back corner */
} Grect3;

    typedef struct {  /* COORDINATE POINT */
      Gfloat    x;    /* X coordinate */
      Gfloat    y;    /* Y coordinate */
      Gfloat    z;    /* Z coordinate */
    } Gpoint3;

typedef struct {     /* DIMENSION IN INTEGER VALUES */
  Gint      x_dim;   /* X dimension */
  Gint      y_dim;   /* Y dimension */
} Gidim;
```

## Description

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

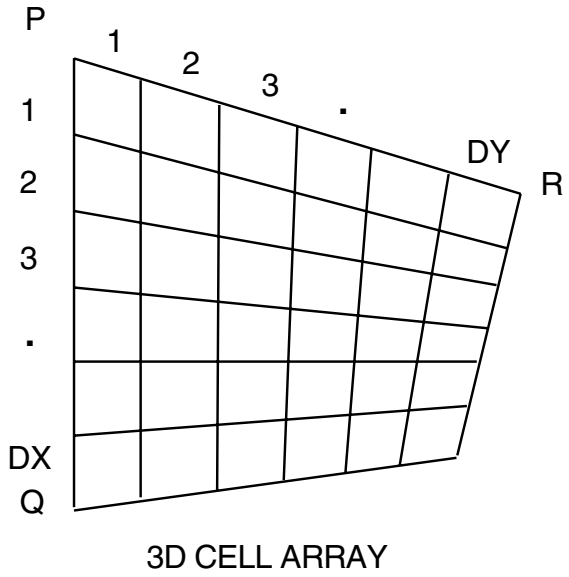
You pass a two-dimensional array containing color index values as one argument to this function. DEC GKS maps the color index values to corresponding cells within a parallelogram-shaped area of the workstation surface. In addition to the color index array, you specify the number of array columns and 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 DX and DY 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.




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

gfillarea (

```
  Gint     npoints,   /* (I) Number of vertices in the polygon. */
  Gpoint   *points    /* (I) X and Y world coordinates. The size of
                             the array is equal to the value of
                             npoints. */
```

)

### Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;
```

### 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

gfillarea3 (

```
  Gint      npoints,  /* (I) Number of points in the polygon. */
  Gpoint3   *points   /* (I) X, Y, and Z world coordinates.  The size
                             of the array is equal to the value of
                             npoints. */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
  Gfloat    z;    /* Z coordinate */
} Gpoint3;
```

## 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

gfillareaset (

```
  Gint    num_lists,     /* (I) Number of fill areas in the set. */
  Gint    *size_list,    /* (I) Array of integers that specifies the number
                                 of points that bound each fill area. */
  Gpoint  *points        /* (I) Array of WC points that bound each fill
                                 area in the set. The size of the array is
                                 equal to the sum of all values in the
                                 array size_list. */
```

)

### Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;
```

### 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

gfillareaset3 (

```
  Gint       num_lists,    /* (I) Number of fill areas in the set. */
  Gint       *size_list,   /* (I) Array of integers that specifies
                                  the number of points that bound each fill
                                  area. */
  Gpoint3    *points       /* (I) Array of WC points that bound each fill
                                  area in the set. The size of the array is
                                  equal to the sum of all values in the
                                  array size_list. */
```

)

### Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
  Gfloat    z;    /* Z coordinate */
} Gpoint3;
```

### 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 (

```
  Gint       npoints,    /* (I) Number of points in the GDP. */
  Gpoint     *points,    /* (I) Array of WC coordinate points.  The
                                size of the array is equal to the
                                value of npoints. */
  Gint       function,   /* (I) GDP function identifier (constant). */
  Ggdprec    *data       /* (I) GDP data record. */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;

typedef union {                       /* GDP DATA RECORD */
  Gugdp_datarec    gdp_datarec;
} Ggdprec;

    typedef struct {                  /* escape-dependent data record */
      Gint      number_integer;       /* number of integer */
      Gint      number_float;         /* number of float */
      Gint      number_strings;       /* number of string */
      Gint      *list_integers;       /* list of integers */
      Gfloat    *list_floats;         /* list of floats */
      Gint      *list_string_lengths; /* list of string lengths */
      Gchar     **list_strings;       /* list of strings */
    } Gugdp_datarec;
```

## Constants

| GDP Identifier | Description |
|---|---|
| GDP_DISP | Disjoint Polyline |
| GDP_CCP | Circle: Center and Point on Circumference |
| GDP_C3P | Circle: Three Points on Circumference |
| GDP_CCR | Circle: Center and Radius |
| GDP_C2PR | Circle: Two Points on Circumference, and Radius |
| GDP_AC2P | Arc: Center and Two Points on Arc |
| GDP_A3P | Arc: Three Points on Circumference |
| GDP_ACVR | Arc: Center, Two Vectors, and a Radius |
| GDP_A2PR | Arc: Two Points on Arc and Radius |
| GDP_ACPA | Arc: Center, Starting Point, and Angle |
| GDP_ECA | Ellipse: Center, and Two Axis Vectors |
| GDP_EFP | Ellipse: Focal Points and Point on Circumference |
| GDP_EACA | Elliptic Arc: Center, Two Axis Vectors, and Two Vectors |
| GDP_EAFP | Elliptic Arc: Focal Points and Two Points on Circumference |

```
GDP_R2P         Rectangle: Two Corners
GDP_FAS         Fill Area Set
GDP_FCCP        Filled Circle: Center and Point on Circumference
GDP_FC3P        Filled Circle: Three Points on Circumference
GDP_FCCR        Filled Circle: Center and Radius
GDP_FCPR        Filled Circle: Two Points on Circumference, and Radius
GDP_FACP        Filled Arc: Center and Two Points on Arc
GDP_FA3P        Filled Arc: Three Points on Circumference
GDP_FACV        Filled Arc: Center, Two Vectors, and a Radius
GDP_FAPR        Filled Arc: Two Points on Arc, and Radius
GDP_FACA        Filled Arc: Center, Starting Point, and Angle
GDP_FECA        Filled Ellipse: Center and Two Axis Vectors
GDP_FEFP        Filled Ellipse: Focal Points and Point on Circumference
GDP_FEACA       Filled Elliptic Arc: Center, Two Axis Vectors, and Two Vectors
GDP_FEAF        Filled Elliptic Arc: Focal Points and Two Points on
                Circumference
GDP_FR2P        Filled Rectangle: Two Corners
GDP_GIA         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 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

Example 5–2 for a program example using the GENERALIZED DRAWING PRIMITIVE function

# GENERALIZED DRAWING PRIMITIVE 3

## Operating States

WSAC, SGOP

## Syntax

ggdp3 (

```
  Gint        npoints,    /* (I) Number of points in the GDP. */
  Gpoint3     *points,    /* (I) Array of WC coordinate points.  The
                                 size of points.  The size of the array
                                 is equal to the value of npoints. */
  Gint        function,   /* (I) GDP function identifier (constant). */
  Ggdprec     *data       /* (I) GDP 3 data record. */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
  Gfloat    z;    /* Z coordinate */
} Gpoint3;

typedef union {    /* GDP DATA RECORD */
  Gugdp_datarec    gdp_datarec;
} Ggdprec;

    typedef struct {                  /* escape-dependent data record */
      Gint      number_integer;       /* number of integers */
      Gint      number_float;         /* number of floats */
      Gint      number_strings;       /* number of strings */
      Gint      *list_integers;       /* list of integers */
      Gfloat    *list_floats;         /* list of floats */
      Gint      *list_string_lengths; /* list of string lengths */
      Gchar     **list_strings;       /* list of strings */
    } Gugdp_datarec;
```

## 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 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

Example 5–2 for a program example using the GENERALIZED DRAWING PRIMITIVE function

# POLYLINE

## Operating States

WSAC, SGOP

## Syntax

gpolyline (

```
  Gint      npoints,    /* (I) Number of points in polyline. */
  Gpoint    *points     /* (I) X and Y world coordinates of the points.
                               The size of the array is equal to the
                               value of npoints. */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;
```

## 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

gpolyline3 (

```
  Gint       npoints,    /* (I) Number of points in the polyline. */
  Gpoint3    *points     /* (I) X, Y, and Z world coordinates of the points.
                                The size of the array is equal to the
                                value of npoints. */
```

)

### Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
  Gfloat    z;    /* Z coordinate */
} Gpoint3;
```

### 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

gpolymarker (

```
  Gint      npoints,    /* (I) Number of polymarker locations. */
  Gpoint    *points     /* (I) X and Y world coordinates of the polymarkers.
                               The size of the array is equal to the
                               value of npoints. */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;
```

## 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

gpolymarker3 (

```
  Gint       npoints,    /* (I) Number of polymarker locations. */
  Gpoint3    *points     /* (I) X, Y, and Z world coordinates of the
                                polymarkers.  The size of the array is equal
                                to the value of npoints. */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
  Gfloat    z;    /* Z coordinate */
} Gpoint3;
```

## 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

gtext (

```
  Gpoint    *position,    /* (I) WC position of text string */
  Gchar     *text        /* (I) String to be written */
```

)

## Data Structures

```
typedef struct  {  /* COORDINATE POINT */
  Gfloat    x;     /* X coordinate */
  Gfloat    y;     /* Y coordinate */
} Gpoint;
```

## 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

gtext3 (

```
  Gpoint3     *position,     /* (I) WC position of text string. */
  Gpoint3     *dirn_vec1,    /* (I) Vector 1 for the text plane definition. */
  Gpoint3     *dirn_vec2,    /* (I) Vector 2 for the text plane definition.
                                    The text plane = vector1×  vector2 */
  Gchar       *text          /* (I) String to be written. */
```

)

## Data Structures

```
typedef struct  {  /* COORDINATE POINT */
  Gfloat    x;     /* X coordinate */
  Gfloat    y;     /* Y coordinate */
  Gfloat    z;     /* Z coordinate */
} Gpoint3;
```

## 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**

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

# include <stdio.h>
# include <gks.h>          /* GKS C binding header file */

# define XDIM 10
# define YDIM 1

main()
{
   Gint       color[XDIM][YDIM];
   Gconn      default_conid;
   Gwstype    default_wstype;
   Gdefmode   defer_mode;
   Gidim      dimensions;
   Gevent     event;
   Gint       i;
   Grect      rectangle;
   Girgmode   regen_mode;
   Gfloat     timeout;
   Gregen     update_flag;
   Gint       ws_id;

/* Open the GKS and workstation environments. */

   default_conid    = GWC_DEF;
   default_wstype   = GWS_DEF;
   defer_mode       = GASAP;
   regen_mode       = GALLOWED;
   ws_id            = 1;

   gopengks (0, 0);
   gopenws (ws_id, &default_conid, &default_wstype);
   gactivatews (ws_id);
   gsetdeferst (ws_id, defer_mode, regen_mode);

/* Initialize the color index array. Draw the stripes. Wait 5 seconds. */

   for ( i = 0; i < 10; i += 2)
     {
     color[i][0]   = 0;
     color[i+1][0] = 1;
     }

   dimensions.x_dim = XDIM;
   dimensions.y_dim = YDIM;
   rectangle.ul.x   = 0.0;
   rectangle.ul.y   = 1.0;
   rectangle.lr.x   = 1.0;
   rectangle.lr.y   = 0.0;
   timeout          = 5.0;
   update_flag      = GPOSTPONE;
```

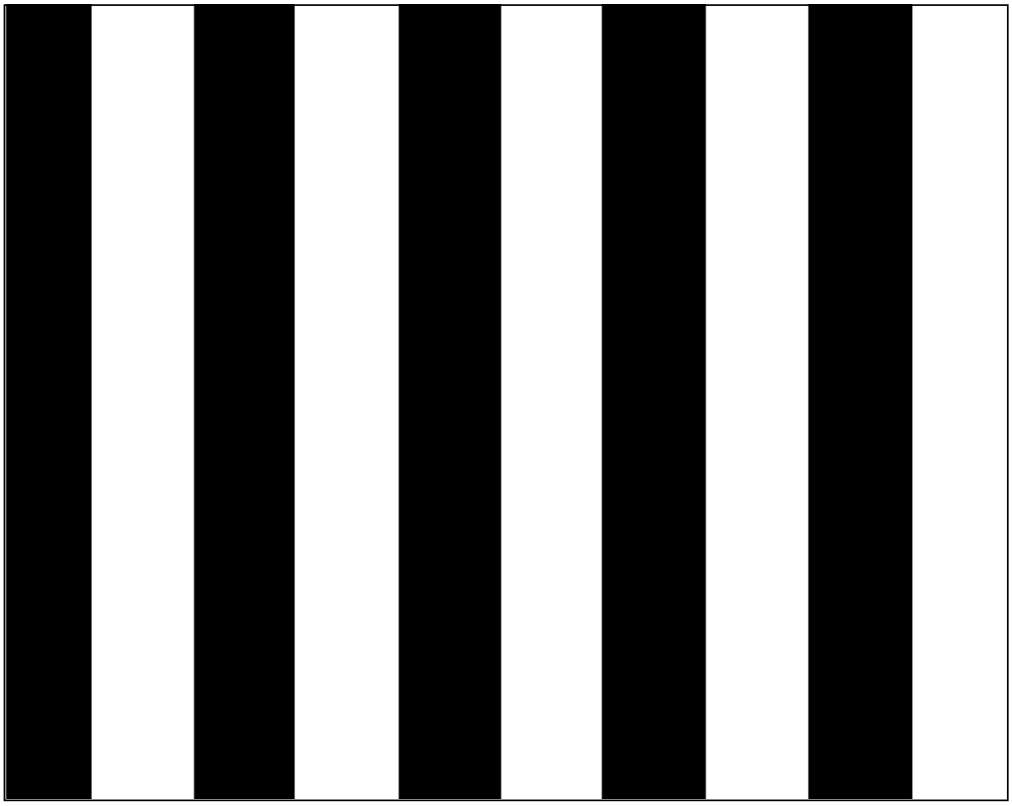
(continued on next page)

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

```
   gcellarray (&rectangle, &dimensions, color);
   gupdatews (ws_id, update_flag);
   gawaitevent (timeout, &event);

/* Release the GKS and workstation environments. */

   gdeactivatews (ws_id);
   gclosews (ws_id);
   gclosegks ();
}
```

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

**Figure 5–1 Cell Array Output**



ZK–4015A–GE

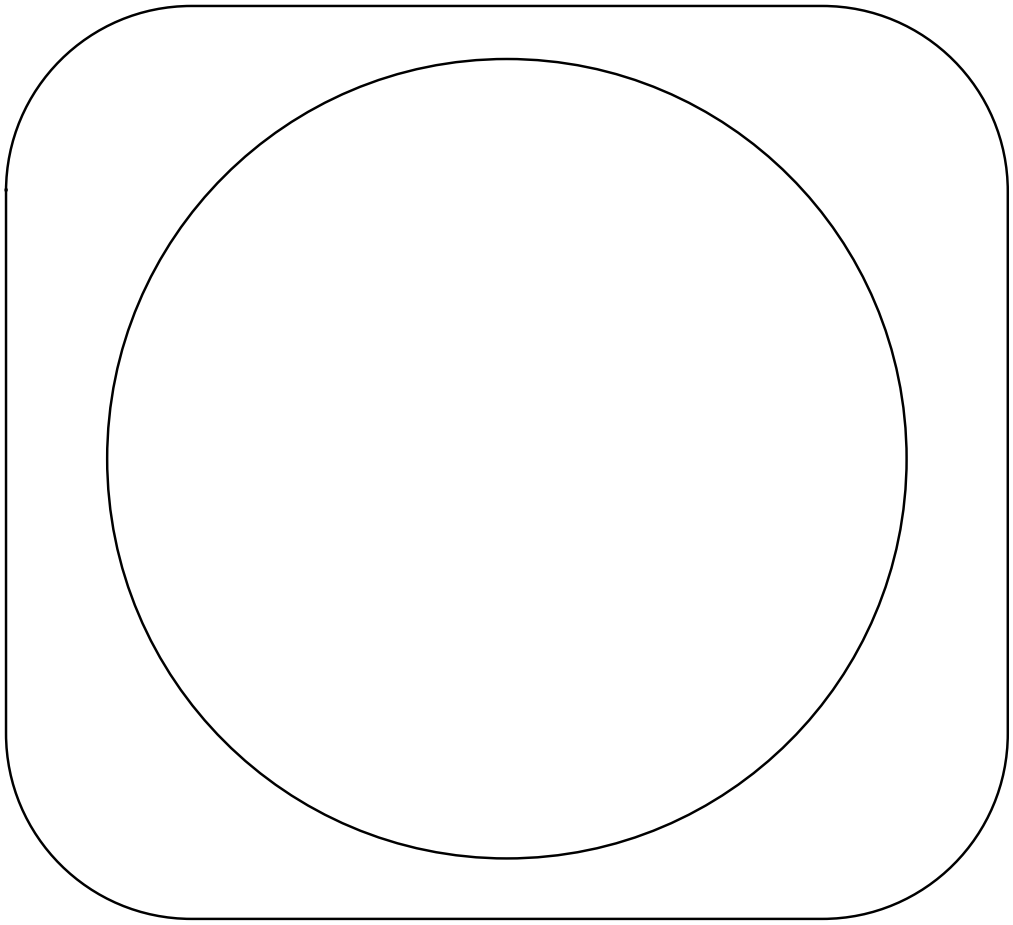
Example 5–2 illustrates the use of the GENERALIZED DRAWING PRIMITIVE function.

**Example 5–2 Generalized Drawing Primitive Output**

```
/*
 *  This program creates an unfilled circle using the GDP
 *  GDP_C3P (-102).
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

# include <stdio.h>
# include <gks.h>          /* C binding definition file */

main ()
{
    Ggdprec   data;
    Gconn     default_conid;
    Gwstype   default_wstype;
    Gevent    event;
    Gint      num_points   = 3;
    Gpoint    points[3];
    Gfloat    timeout      = 5.0;
    Gint      ws_id        = 1;

/*  Open the GKS and workstation environments. */

    default_conid = GWC_DEF;
    default_wstype = GWS_DEF;

    gopengks (0, 0);
    gopenws (ws_id, &default_conid, &default_wstype);
    gactivatews (ws_id);

/*
 *  The constant GDP_C3P specifies the GDP identification number -102.
 *  This GDP creates a circle using three points on a circle's
 *  circumference.  This particular GDP does not require a data record
 *  to perform its task.  Notice that DEC GKS uses the current polyline
 *  attributes to create the circle.
 */

    points[0].x    = 0.1; points[0].y = 0.5;
    points[1].x    = 0.5; points[1].y = 0.1;
    points[2].x    = 0.9; points[2].y = 0.5;

    ggdp (num_points, points, GDP_C3P, &data);
    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout, &event);

/*  Release the GKS and workstation environments. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();
}
```

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 data structure with 13 fields, one field for each nongeometric attribute. Each field contains either the value GBUNDLED (0) or the value GINDIVIDUAL (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 GBUNDLED 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
INQUIRE CURRENT PRIMITIVE ATTRIBUTE VALUES
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

gsetasf (

```
  Gasfs  *flags   /* (I) Pointer to aspect source flags */
```

)

## Data Structures

```
typedef struct {         /* ASPECT SOURCE FLAGS (constants) */
  Gasf      ln_type;     /* line type */
  Gasf      ln_width;    /* line width */
  Gasf      ln_colour;   /* line color */
  Gasf      mk_type;     /* marker type */
  Gasf      mk_size;     /* marker size */
  Gasf      mk_colour;   /* marker color */
  Gasf      tx_fp;       /* text font and precision */
  Gasf      tx_exp;      /* text expansion */
  Gasf      tx_space;    /* text character spacing */
  Gasf      tx_colour;   /* text color */
  Gasf      fl_inter;    /* fill area interior style */
  Gasf      fl_style;    /* fill area style index */
  Gasf      fl_colour;   /* fill area color */
} Gasfs;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gasf | GBUNDLED | Bundled attributes |
| | GINDIVIDUAL | Individual attributes |

**Note**

You must set all the ASFs before making the call.

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

The initial value for each ASF is GINDIVIDUAL, 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

gsetasf3 (

```
  Gasfs3   *flags    /* (I) Aspect source flags */
```

)

## Data Structures

```
typedef struct {          /* ASPECT SOURCE FLAGS 3 (constants) */
  Gasf      ln_type;      /* line type */
  Gasf      ln_width;     /* line width */
  Gasf      ln_colour;    /* line color */
  Gasf      mk_type;      /* marker type */
  Gasf      mk_size;      /* marker size */
  Gasf      mk_colour;    /* marker color */
  Gasf      tx_fp;        /* text font and precision */
  Gasf      tx_exp;       /* text expansion */
  Gasf      tx_space;     /* text character spacing */
  Gasf      tx_colour;    /* text color */
  Gasf      fl_inter;     /* fill area interior style */
  Gasf      fl_style;     /* fill area style index */
  Gasf      fl_colour;    /* fill area color */
  Gasf      edge_flag;    /* edge flag asf */
  Gasf      edge_type;    /* edge type asf */
  Gasf      edge_width;   /* edge width asf */
  Gasf      edge_colour;  /* edge color asf */
} Gasfs3;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gasf | GBUNDLED | Bundled attributes |
| | GINDIVIDUAL | Individual attributes |

**Note**

You must set all the ASFs before making the call.

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

The initial value for each ASF is GINDIVIDUAL, 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

gsetcharexpan (

```
  Gfloat   exp   /* (I) Character width 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

gsetcharheight (

```
  Gfloat    height    /* (I) Character height in WC values.  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

gsetcharspace (

```
  Gfloat  spacing   /* (I) Spacing as a percentage of 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

gsetcharup (

```
  Gpoint  *charup     /* (I) Pointer to vector X, Y values that
                             specify the slope of the character up
                             vector. The default value is (0.0, 1.0). */
```

)

### Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;
```

### 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

gsetcolourmodel (

```
  Gint         ws,       /* (I) Workstation identifier */
  Gcolmodel    model     /* (I) Color model (constant) */
```

)

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcolmodel | GCM_RGB | Red, green, and blue color model |
| | GCM_CIE | Commission Internationale de L'Eclairage color model |
| | GCM_HSV | Hue, saturation, and value color model |
| | GCM_HLS | 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

gsetcolourrep (

```
  Gint      ws,      /* (I) Workstation identifier */
  Gint      index,   /* (I) Color index */
  Gcobundl  *rep     /* (I) Pointer to color bundle values.  This is a
                            color triplet that corresponds to the current
                            color model. */
```

)

### Data Structures

```
typedef struct {     /* COLOR BUNDLE */
  Gfloat    red;     /* red intensity */
  Gfloat    green;   /* green intensity */
  Gfloat    blue;    /* blue intensity */
} Gcobundl;
```

### 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. |

| Model | Description | Values |
|---|---|---|
| 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

gsetedgecolourind (

```
  Gint     colour     /* (I) 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

gsetedgeflag (

```
  Gedge_f       flag         /* (I) Edge flag (constant) */
```

)

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gedge_f | GEDGE_OFF | Edge off. This is the default value. |
| | GEDGE_ON | 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

gsetedgeindex (

```
  Gint   index    /* (I) 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

gsetedgerep (

```
  Gint         ws,       /* (I) Workstation identifier */
  Gint         index,    /* (I) Edge index */
  Gedgebundl   *rep      /* (I) Pointer to edge bundle values */
```

)

## Data Structures

```
typedef struct {      /* EDGE BUNDLE */
  Gedge_f   flag;     /* edge flag (constant) */
  Gint      type;     /* edge type (constant) */
  Gfloat    width;    /* edge width scale factor */
  Gint      colour;   /* edge color index */
} Gedgebundl;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gedge_f | GEDGE_OFF | Edge off. This is the default value. |
| | GEDGE_ON | Edge on. |
| Edge types | GED_SOLID | Solid edge. This is the default value. |
| | GED_DASHED | Dashed edge. |
| | GED_DOTTED | Dotted edge. |
| | GED_DASHDOT | 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

gsetedgetype (

```
  Gint    type    /* (I) Edge type (constant) */
```

)

### Constants

| Defined Argument | Constant | Description |
|---|---|---|
| type | GED_SOLID | Solid edge. This is the default value. |
| | GED_DASHED | Dashed edge. |
| | GED_DOTTED | Dotted edge. |
| | GED_DASHDOT | 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

gsetedgewidthscfac (

```
  Gfloat   width    /* (I) 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

gsetfillcolourind (

```
  Gint  colour    /* (I) 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

gsetfillind (

```
  Gint  index    /* (I) 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

gsetfillintstyle (

```
  Gflinter    style   /* (I) Interior style (constant) */
```

)

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gflinter | GHOLLOW | Hollow interior. This is the default value. |
| | GSOLID | Solid interior. |
| | GPATTERN | 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

gsetfillrep (

```
  Gint        ws,       /* (I) Workstation identifier */
  Gint        index,    /* (I) Fill area bundle index */
  Gflbundl    *rep      /* (I) Pointer to bundle table values */
```

)

## Data Structures

```
typedef struct {     /* FILL AREA BUNDLE */
  Gflinter  inter;   /* fill area interior style (constant) */
  Gint      style;   /* fill area style index */
  Gint      colour;  /* fill area color index */
} Gflbundl;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gflinter | GHOLLOW | Hollow interior. This is the default value. |
| | GSOLID | Solid interior. |
| | GPATTERN | Patterned interior. |
| | GHATCH | Hatched interior. |

**Note**

If GHOLLOW or GSOLID are specified for the interior style field, DEC GKS ignores the style index field.

## 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

gsetfillstyleind (

```
  Gint   index    /* (I) 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

gsethlhsrid (

```
  Gint    hlhsrid    /* (I) HLHSR identifier (constant) */
```

)

### Constants

| Defined Argument | Constant | Description |
|---|---|---|
| hlhsrid | GHLHSR_ID_NONE | No HLHSR processing. This is the default value. |
| | GHLHSR_ID_PAINTER | 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

gsethlhsrmode (

```
  Gint    ws,      /* (I) Workstation identifier */
  Gint    mode     /* (I) HLHSR mode (constant) */
```

)

### Constants

| Defined Argument | Constant | Description |
|---|---|---|
| mode | GHLHSR_MODE_NONE | No HLHSR processing. This is the default value. |
| | GHLHSR_MODE_PAINTER | 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

gsetlinetype (

```
  Gint    type    /* (I) Polyline type (constant) */
```

)

### Constants

| Defined Argument | Constant | Description |
|---|---|---|
| type | GLN_SOLID | Solid line. This is the default value. |
| | GLN_DASHED | Dashed line. |
| | GLN_DOTTED | Dotted line. |
| | GLN_DASHDOT | 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

gsetlinewidth (

```
  Gfloat   width    /* (I) 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

gsetmarkersize (

```
  Gfloat  width     /* (I) 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

gsetmarkertype (

```
  Gint    type    /* (I) Marker type (constant) */
```

)

## Constants

| Defined Argument | Constant | Description |
|---|---|---|
| type | GMK_POINT | Dot. |
| | GMK_PLUS | Plus sign. |
| | GMK_STAR | Asterisk. This is the default value. |
| | GMK_O | Circle. |
| | GMK_X | 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

gsetpatrefpt (

```
  Gpoint  *patref   /* (I) WC pattern start point X and Y */
```

)

### Data Structures

```
typedef struct {    /* COORDINATE POINT */
  Gfloat    x;      /* X coordinate */
  Gfloat    y;      /* Y coordinate */
} Gpoint;
```

### 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

gsetpatrefptvec (

```
  Gpoint3   *patref,     /* (I) 3D reference point */
  Gpoint3   *refvec1,    /* (I) WC pattern reference vector 1 */
  Gpoint3   *refvec2     /* (I) WC pattern reference vector 2 */
```

)

### Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
  Gfloat    z;    /* Z coordinate */
} Gpoint3;
```

### 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

gsetpatrep (

```
  Gint        ws,      /* (I) Workstation identifier */
  Gint        index,   /* (I) Pattern index */
  Gptbundl    *rep     /* (I) Pointer to pattern bundle values */
```

)

### Data Structures

```
typedef struct {      /* PATTERN BUNDLE */
  Gipoint   size;     /* pattern array size */
  Gint      *array;   /* pattern array */
} Gptbundl;

    typedef struct {  /* INTEGER POINT */
      Gint      x;    /* X coordinate */
      Gint      y;    /* Y coordinate */
    } Gipoint;
```

### 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

gsetpatsize (

```
  Gpoint    *patsize    /* (I) Pointer to X and Y sizes, in WC values */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;
```

## 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

gsetpickid (

```
  Gint    pickid     /* (I) 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

gsetlinecolourind (

```
  Gint    colour    /* (I) 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

gsetlineind (

```
  Gint   index    /* (I) Polyline bundle 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

gsetlinerep (

```
  Gint        ws,        /* (I) Workstation identifier */
  Gint        index,     /* (I) Polyline bundle index */
  Glnbundl    *rep       /* (I) Pointer to bundle table values */
```

)

## Data Structures

```
typedef struct {         /* POLYLINE BUNDLE */
  Gint      type;        /* line type (constant) */
  Gfloat    width;       /* line width scale factor */
  Gint      colour;      /* polyline color index */
} Glnbundl;
```

## Constants

| Data Structure Constant | Constant | Description |
|---|---|---|
| Line types | GLN_SOLID | Solid line. This is the default value. |
| | GLN_DASHED | Dashed line. |
| | GLN_DOTTED | Dotted line. |
| | GLN_DASHDOT | 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

gsetmarkercolourind (

```
  Gint   colour    /* (I) 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

gsetmarkerind (

```
  Gint   index    /* (I) Polymarker bundle 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

gsetmarkerrep (

```
  Gint       ws,       /* (I) Workstation identifier */
  Gint       index,    /* (I) Polymarker bundle index */
  Gmkbundl   *rep      /* (I) Pointer to bundle table values */
```

)

## Data Structures

```
typedef struct {       /* POLYMARKER BUNDLE */
  Gint      type;      /* marker type (constant) */
  Gfloat    size;      /* marker size scale factor */
  Gint      colour;    /* polymarker color index */
} Gmkbundl;
```

## Constants

| Data Structure Constant | Constant | Description |
|---|---|---|
| Marker types | GMK_POINT | Dot. |
| | GMK_PLUS | Plus sign. |
| | GMK_STAR | Asterisk. This is the default value. |
| | GMK_O | Circle. |
| | GMK_X | 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

gsettextalign (

```
  Gtxalign  *txalign   /* (I) Pointer to horizontal and vertical
                             alignments */
```

)

### Data Structures

```
typedef struct {       /* TEXT ALIGNMENT */
  Gtxhor    hor;       /* horizontal component (constant) */
  Gtxver    ver;       /* vertical component (constant) */
} Gtxalign;
```

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gtxhor | GAH_NORMAL | Normal horizontal alignment. This is the default value. |
| | GAH_LEFT | Left horizontal alignment. |
| | GAH_CENTRE | Center horizontal alignment. |
| | GAH_RIGHT | Right horizontal alignment. |
| Gtxver | GAV_NORMAL | Normal vertical alignment. This is the default value. |
| | GAV_TOP | Top vertical alignment. |
| | GAV_CAP | Cap vertical alignment. |
| | GAV_HALF | Half vertical alignment. |
| | GAV_BASE | Base vertical alignment. |
| | GAV_BOTTOM | 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

gsettextcolourind (

```
  Gint  colour    /* (I) Text color representation 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

gsettextfontprec (

```
  Gtxfp  *txfp    /* (I) Pointer to font and precision values */
```

)

### Data Structures

```
typedef struct {    /* TEXT FONT AND PRECISION */
  Gint      font;   /* text font-- default value is 1 */
  Gtxprec   prec;   /* text precision (constant) */
} Gtxfp;
```

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gtxprec | GP_STRING | String precision. DEC GKS evaluates character height and width attributes only; this is the default precision value. |
| | GP_CHAR | Character precision. DEC GKS evaluates each character for compliance with all other specified text attributes. |
| | GP_STROKE | 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

gsettextind (

```
  Gint   index    /* (I) 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

gsettextpath (

```
  Gtxpath     path      /* (I) Text path (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gtxpath | GTP_RIGHT | Text string reads from left to right. This is the default value. |
| | GTP_LEFT | Text string reads from right to left. |
| | GTP_UP | Text string reads from bottom to top. |
| | GTP_DOWN | 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

gsettextrep (

```
  Gint        ws,       /* (I) Workstation identifier */
  Gint        index,    /* (I) Text bundle index */
  Gtxbundl    *rep      /* (I) Pointer to bundle table values */
```

)

## Data Structures

```
typedef struct {        /* TEXT BUNDLE */
  Gtxfp     fp;         /* font and precision */
  Gfloat    exp;        /* character expansion */
  Gfloat    space;      /* character spacing */
  Gint      colour;     /* text color */
} Gtxbundl;

    typedef struct {    /* TEXT FONT AND PRECISION */
      Gint      font;   /* text font */
      Gtxprec   prec;   /* text precision (constant) */
    } Gtxfp;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gtxprec | GP_STRING | String precision. DEC GKS evaluates character height and width attributes only; this is the default precision value. |
| | GP_CHAR | Character precision. DEC GKS evaluates each character for compliance with all other specified text attributes. |
| | GP_STROKE | 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**

```
/*
 *  This program calls the SET COLOUR REPRESENTATION function to change
 *  the color representation of a particular index from color1 to color2.
 *  This program assumes an RGB color model. To avoid making such an
 *  assumption, use the SET COLOUR MODEL function to set the color model
 *  to RGB explicity.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

# include <stdio.h>
# include <gks.h>       /* GKS C binding definition file */

main ()
{
    Gint      color1       = 4;
    Gcobundl  color2_rep;
    Gconn     conn_id      = GWC_DEF;
    Gevent    event;
    Gint      figure       = 1;
    Gint      npoints      = 20;
    Gpoint    points[20];
    Gfloat    timeout      = 5.00;
    Gint      ws_id        = 1;
    Gwstype   ws_type      = GWS_DEF;

/*  Open the GKS and workstation environments. */

    gopengks (0, 0);
    gopenws (ws_id, &conn_id, &ws_type);
    gactivatews (ws_id);

/*
 *  Store the figure in a segment. Set the fill index to color1, set
 *  the fill interior style to SOLID, initialize the figure, and draw the
 *  figure.
 */

    points[0].x = 0.1;    points[0].y = 0.1;
    points[1].x = 0.4;    points[1].y = 0.1;
    points[2].x = 0.4;    points[2].y = 0.2;
    points[3].x = 0.6;    points[3].y = 0.2;
    points[4].x = 0.6;    points[4].y = 0.1;
    points[5].x = 0.9;    points[5].y = 0.1;
    points[6].x = 0.9;    points[6].y = 0.4;
    points[7].x = 0.8;    points[7].y = 0.4;
    points[8].x = 0.8;    points[8].y = 0.6;
    points[9].x = 0.9;    points[9].y = 0.6;
```

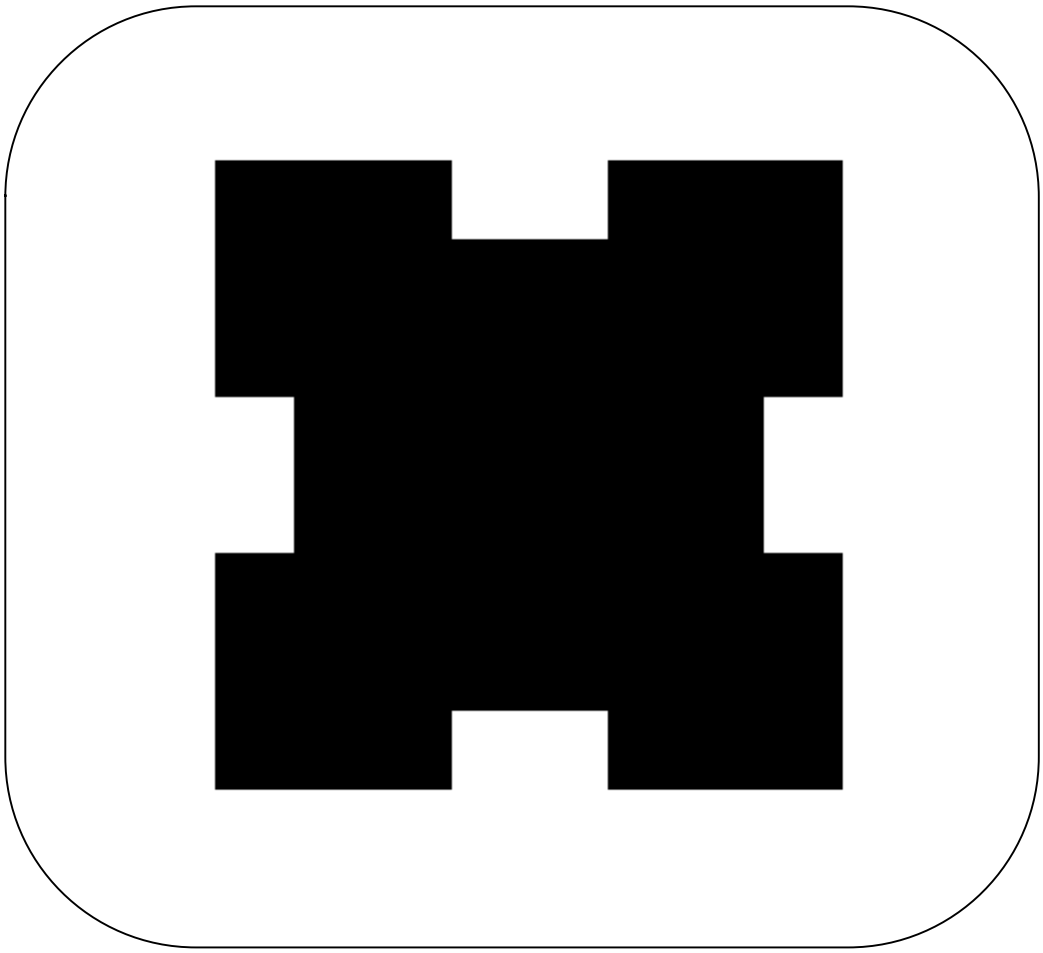
(continued on next page)

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

```
    points[10].x = 0.9;   points[10].y = 0.9;
    points[11].x = 0.6;   points[11].y = 0.9;
    points[12].x = 0.6;   points[12].y = 0.8;
    points[13].x = 0.4;   points[13].y = 0.8;
    points[14].x = 0.4;   points[14].y = 0.9;
    points[15].x = 0.1;   points[15].y = 0.9;
    points[16].x = 0.1;   points[16].y = 0.6;
    points[17].x = 0.2;   points[17].y = 0.6;
    points[18].x = 0.2;   points[18].y = 0.4;
    points[19].x = 0.1;   points[19].y = 0.4;

    gcreateseg (figure);
    gsetfillcolourind (color1);
    gsetfillintstyle (GSOLID);
    gfillarea (npoints, points);
    gcloseseg ();


/*  Wait 5 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout, &event);

/*
 *  Change the color representation of color1 to color2.
 *  This will change the fill color of the figure from color1 to
 *  color2.  Wait 5 seconds.
 */

    color2_rep.comp1 = 1.0;
    color2_rep.comp2 = 0.43;
    color2_rep.comp3 = 0.09;

    gsetcolourrep (ws_id, color1, &color2_rep);
    gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);

/*  Close the GKS and workstation environments. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();
}
```

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**

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

# include <stdio.h>
# include <gks.h>          /* GKS C binding definitions file */
```

(continued on next page)

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

```
main ()
{
    Gconn     conn_id     = GWC_DEF;
    Gevent    event;
    Gint      figure      = 1;
    Gflbundl  fill_rep;
    Gasfs     flags;
    Gint      index       = 6;
    Gint      npoints     = 20;
    Gpoint    points[20];
    Gfloat    timeout     = 5.00;
    Gint      ws_id       = 1;
    Gwstype   ws_type     = GWS_DEF;

/*  Open the GKS and workstation environments. */

    gopengks (0, 0);
    gopenws (ws_id, &conn_id, &ws_type);
    gactivatews (ws_id);

/*  Set the attribute source flags (ASFs) to BUNDLED. */

    flags.ln_type = GBUNDLED;
    flags.ln_width = GBUNDLED;
    flags.ln_colour = GBUNDLED;
    flags.mk_type = GBUNDLED;
    flags.mk_size = GBUNDLED;
    flags.mk_colour = GBUNDLED;
    flags.tx_fp = GBUNDLED;
    flags.tx_exp = GBUNDLED;
    flags.tx_space = GBUNDLED;
    flags.tx_colour = GBUNDLED;
    flags.fl_inter = GBUNDLED;
    flags.fl_style = GBUNDLED;
    flags.fl_colour = GBUNDLED;

    gsetasf (&flags);

/*
 *  Put all output in a segment, initialize the figure,
 *  and draw the figure using the fill area corresponding
 *  to the index value 6.
 */
```

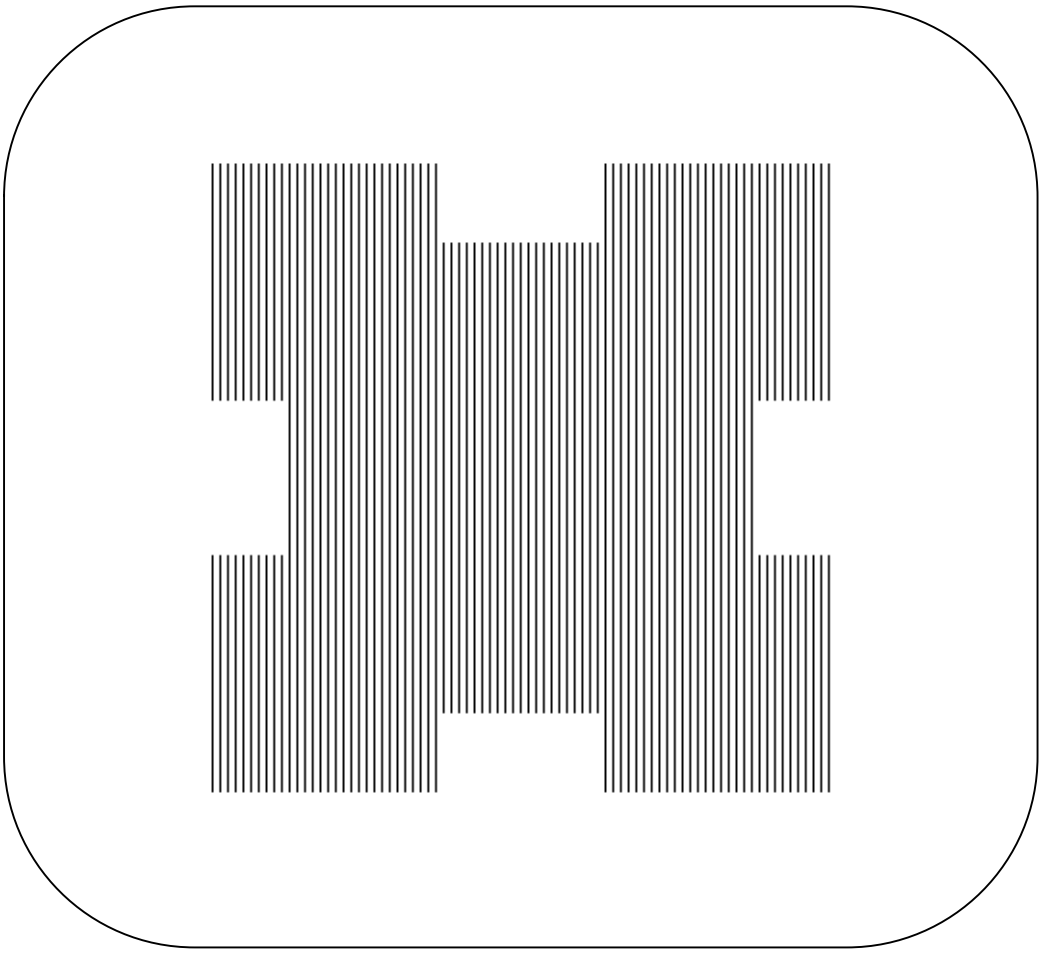
(continued on next page)

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

```
    points[0].x = 0.1;    points[0].y = 0.1;
    points[1].x = 0.4;    points[1].y = 0.1;
    points[2].x = 0.4;    points[2].y = 0.2;
    points[3].x = 0.6;    points[3].y = 0.2;
    points[4].x = 0.6;    points[4].y = 0.1;
    points[5].x = 0.9;    points[5].y = 0.1;
    points[6].x = 0.9;    points[6].y = 0.4;
    points[7].x = 0.8;    points[7].y = 0.4;
    points[8].x = 0.8;    points[8].y = 0.6;
    points[9].x = 0.9;    points[9].y = 0.6;
    points[10].x = 0.9;   points[10].y = 0.9;
    points[11].x = 0.6;   points[11].y = 0.9;
    points[12].x = 0.6;   points[12].y = 0.8;
    points[13].x = 0.4;   points[13].y = 0.8;
    points[14].x = 0.4;   points[14].y = 0.9;
    points[15].x = 0.1;   points[15].y = 0.9;
    points[16].x = 0.1;   points[16].y = 0.6;
    points[17].x = 0.2;   points[17].y = 0.6;
    points[18].x = 0.2;   points[18].y = 0.4;
    points[19].x = 0.1;   points[19].y = 0.4;

    gcreateseg (figure);
    gsetfillind (index);
    gfillarea (npoints, points);
    gcloseseg ();

/*  Wait 5 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout, &event);

/*  Change the attributes associated with fill area index 6. */

    fill_rep.colour = 1;
    fill_rep.inter = GHATCH;
    fill_rep.style = -9;

    gsetfillrep (ws_id, index, &fill_rep);

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

    gupdatews (ws_id, GPERFORM);
    gawaitevent (timeout, &event);

/*  Close the GKS and workstation environments. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();
}
```

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**

```
/*
 *  This program calls the SET LINETYPE function to set the
 *  line type to the dashed and dotted line. The program
 *  draws a line figure displaying the set line type.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

# include <stdio.h>
# include <gks.h>       /* GKS C binding definitions file */
```

(continued on next page)

**Example 6–3 (Cont.) SET LINETYPE Function**

```
main ()
{
    Gconn     conn_id     = GWC_DEF;
    Gevent    event;
    Gint      npoints     = 29;
    Gpoint    points[29];
    Gfloat    timeout     = 5.00;
    Gint      ws_id       = 1;
    Gwstype   ws_type     = GWS_DEF;

/*  Open the GKS and workstation environments. */

    gopengks (0, 0);
    gopenws (ws_id, &conn_id, &ws_type);
    gactivatews (ws_id );


/*
 *  Set the linetype to dashed and dotted lines, initialize
 *  the line figure, and draw it.
 */

    points[0].x = 0.4;    points[0].y = 0.9;
    points[1].x = 0.1;    points[1].y = 0.9;
    points[2].x = 0.1;    points[2].y = 0.6;
    points[3].x = 0.2;    points[3].y = 0.6;
    points[4].x = 0.2;    points[4].y = 0.4;
    points[5].x = 0.1;    points[5].y = 0.4;
    points[6].x = 0.1;    points[6].y = 0.1;
    points[7].x = 0.4;    points[7].y = 0.1;
    points[8].x = 0.4;    points[8].y = 0.2;
    points[9].x = 0.6;    points[9].y = 0.2;
    points[10].x = 0.6;   points[10].y = 0.1;
    points[11].x = 0.9;   points[11].y = 0.1;
    points[12].x = 0.9;   points[12].y = 0.4;
    points[13].x = 0.8;   points[13].y = 0.4;
    points[14].x = 0.8;   points[14].y = 0.6;
    points[15].x = 0.9;   points[15].y = 0.6;
    points[16].x = 0.9;   points[16].y = 0.9;
    points[17].x = 0.6;   points[17].y = 0.9;
    points[18].x = 0.6;   points[18].y = 0.8;
    points[19].x = 0.3;   points[19].y = 0.8;
    points[20].x = 0.3;   points[20].y = 0.3;
    points[21].x = 0.7;   points[21].y = 0.3;
    points[22].x = 0.7;   points[22].y = 0.7;
    points[23].x = 0.4;   points[23].y = 0.7;
    points[24].x = 0.4;   points[24].y = 0.4;
    points[25].x = 0.6;   points[25].y = 0.4;
    points[26].x = 0.6;   points[26].y = 0.6;
    points[27].x = 0.5;   points[27].y = 0.6;
    points[28].x = 0.5;   points[28].y = 0.5;

    gsetlinetype (GLN_DASHDOT);
    gpolyline (npoints, points);
```

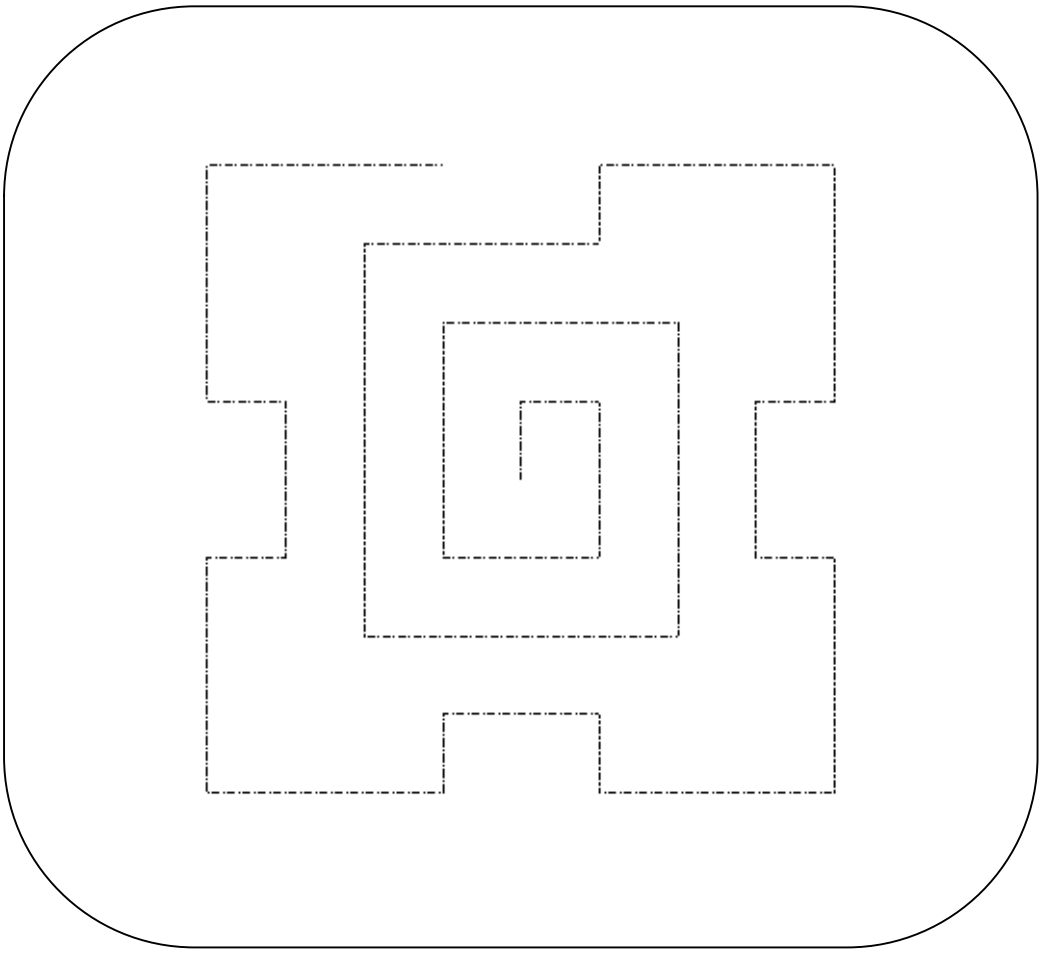
(continued on next page)

**Example 6–3 (Cont.) SET LINETYPE Function**

```
/*  Wait 5 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout, &event);

/*  Close the GKS and workstation environments. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();
}
```

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**

```
/*
 *  This program calls the SET TEXT ALIGNMENT function to write a
 *  string to the workstation using the normal text alignments for
 *  each of four text paths.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

# include <stdio.h>
# include <gks.h>       /* GKS C binding definitions file */

#define TEXT_STRING1 " TEXT LINE 1"
#define TEXT_STRING2 " TEXT LINE 2"
#define TEXT_STRING3 " TEXT LINE 3"
#define TEXT_STRING4 " TEXT LINE 4"

main ()
{
    Gconn     conn_id     = GWC_DEF;
    Gevent    event;
    Gfloat    larger      = 0.07;
    Gint      one_pmark   = 1;
    Gpoint    points[1];
    Gint      red         = 2;
    Gfloat    timeout1    = 1.00;
    Gfloat    timeout2    = 5.00;
    Gtxalign  txalign;
    Gint      ws_id       = 1;
    Gwstype   ws_type     = GWS_DEF;

/*  Open the GKS and workstation environments. */

    gopengks (0, 0);
    gopenws (ws_id, &conn_id, &ws_type);
    gactivatews (ws_id);

/*  Initialize the starting point for all the lines of text. */

    points[0].x = 0.5;
    points[0].y = 0.5;


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

    gsetmarkercolourind (red);
    gsetmarkertype (GMK_PLUS);
    gpolymarker (one_pmark, points);
    gupdatews (ws_id, GPOSTPONE);

/*  Set the text character height and the text alignment. */

    txalign.hor = GAH_NORMAL;
    txalign.ver = GAV_NORMAL;

    gsetcharheight (larger);
    gsettextalign (&txalign);
```

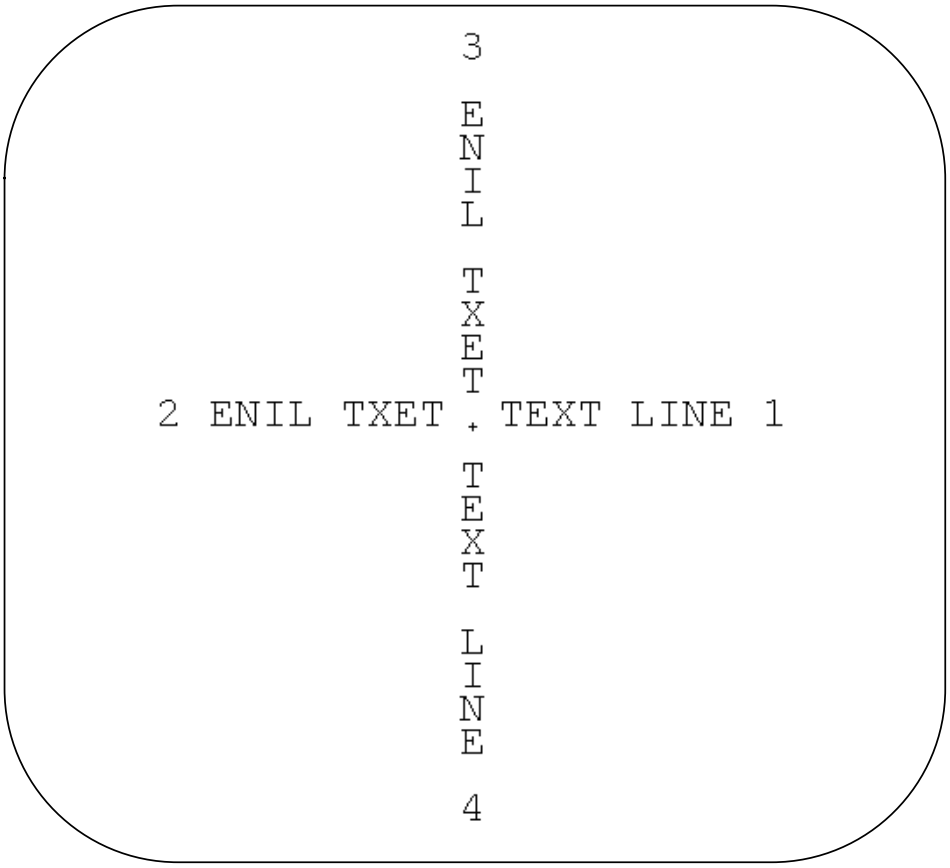
(continued on next page)

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

```
/*
 *  Set a rightward text path and write a character string.
 *  Wait 1 second.
 */

    gsettextpath (GTP_RIGHT);
    gtext (points, TEXT_STRING1);
    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout1, &event);

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

    gsettextpath (GTP_LEFT);
    gtext (points, TEXT_STRING2);
    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout1, &event);


/*
 *  Set an upward text path and write a character string.
 *  Wait 1 second.
 */

    gsettextpath (GTP_UP);
    gtext (points, TEXT_STRING3);
    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout1, &event);

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

    gsettextpath (GTP_DOWN);
    gtext (points, TEXT_STRING4);
    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (timeout2, &event);

/*  Close the GKS and workstation environments. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();
}
```

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.**

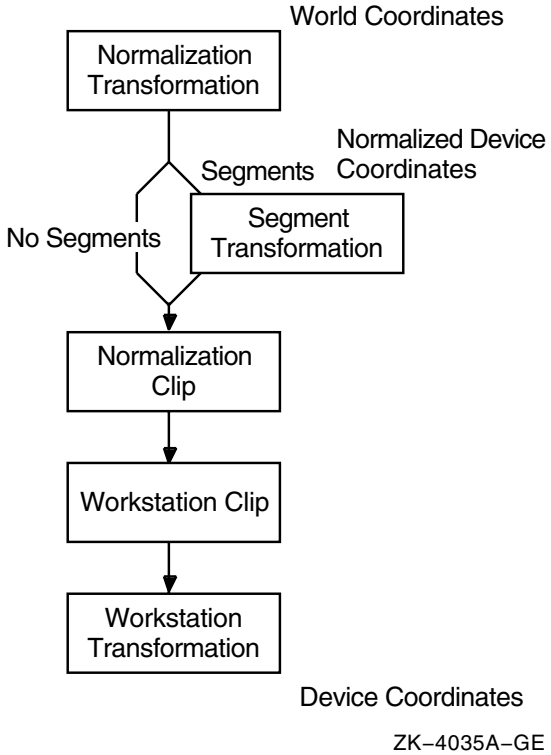
# 7
# Transformation Functions

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.

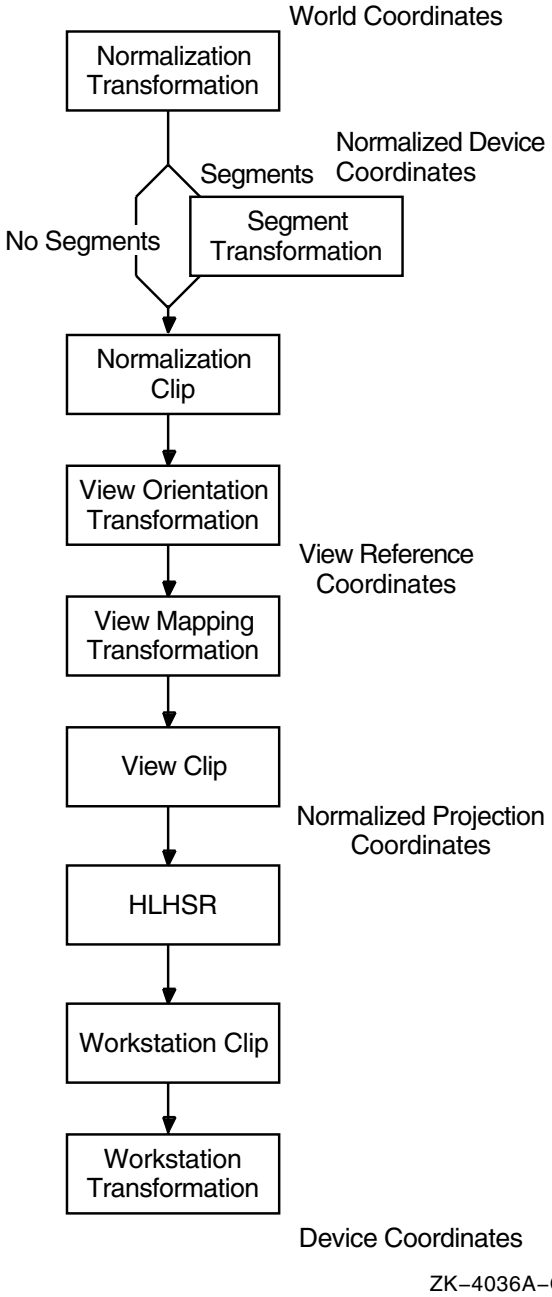
**Figure 7–1 The DEC GKS Two-Dimensional Transformation Pipeline**




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 NDC 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 volumetric 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.

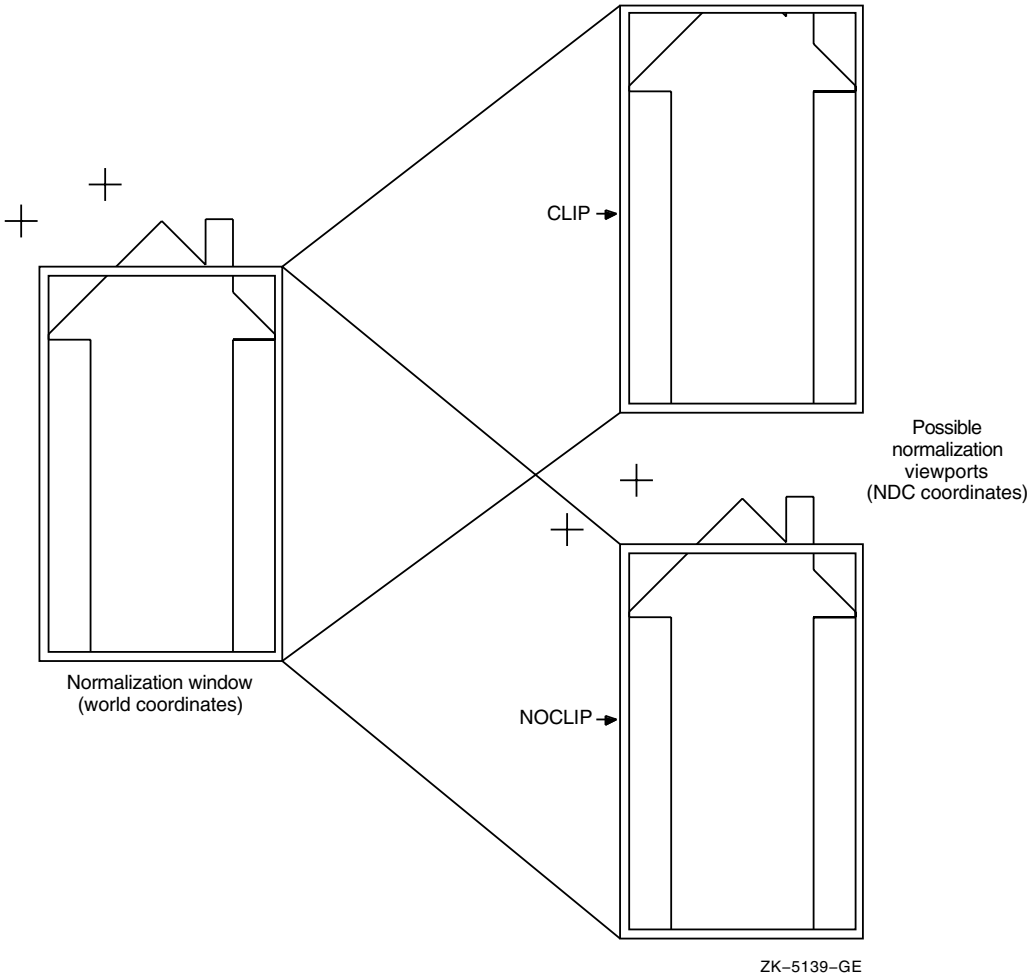
When DEC GKS transforms an image from the normalization window to the NDC plane, there must be a corresponding volume on which to map the contents of the window. This volume portion of the NDC space is called the normalization **viewport**. The two-dimensional default viewport has the range ([0,1] x [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 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 on the NDC plane, 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 on the NDC plane. You must consider the effects this has during input requests. Viewport input priority does not affect output; the order of the output function calls determines which primitive overwrites the other. If you are working with segments, the segment priorities affect overlapping segments. (For more information 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
INQUIRE CLIPPING 3
INQUIRE CURRENT NORMALIZATION TRANSFORMATION NUMBER
INQUIRE DISPLAY SPACE SIZE
INQUIRE DISPLAY SPACE SIZE 3
INQUIRE LIST OF NORMALIZATION TRANSFORMATION NUMBERS
INQUIRE MAXIMUM NORMALIZATION TRANSFORMATION
INQUIRE NORMALIZATION TRANSFORMATION
INQUIRE NORMALIZATION TRANSFORMATION 3
INQUIRE WORKSTATION TRANSFORMATION
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

gaccumtran (

```
  Gfloat    segtran[2][3],   /* (I) Input segment transformation matrix,
                                    created previously by a call to
                                    either EVALUATE TRANSFORMATION
                                    MATRIX or ACCUMULATE TRANSFORMATION
                                    MATRIX. */
  Gpoint    *point,          /* (I) Fixed point for rotation. */
  Gpoint    *shift,          /* (I) Shift vector. Pass the value 0.0 to
                                    avoid translating the segment. */
  Gfloat    angle,           /* (I) Angle of rotation, in radians
                                    (360 degrees = 2*pi radians).  Pass
                                    the value 0.0 to avoid rotating the
                                    segment. */
  Gscale    *scale,          /* (I) X and Y scale factors.  Pass the
                                    value 1.0 to avoid scaling the
                                    segment. */
  Gcsw      coord,           /* (I) WC or NDC units switch (constant). */
  Gfloat    result[2][3]     /* (O) Output segment transform matrix that
                                    results from the concatenation of
                                    the new scaling, rotation, and
                                    translation component values with
                                    the argument segtran. */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;

typedef struct {  /* SCALE VECTOR */
  Gfloat    x_scale;
  Gfloat    y_scale;
} Gscale;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcsw | GWC | The fixed point and shift vectors are WC values. |
| | GNDC | The fixed point and shift vectors are NDC values. |

## 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

gaccumtran3 (

```
  Gfloat     segtran[3][4],  /* (I) Input segment transformation matrix,
                                    created previously by a call to
                                    either EVALUATE TRANSFORMATION
                                    MATRIX 3 or ACCUMULATE TRANSFORMATION
                                    MATRIX 3. */
  Gpoint3    *point,         /* (I) Fixed point for rotation. */
  Gpoint3    *shift,         /* (I) Shift vector.  Pass the value 0.0 to
                                    avoid translating the segment. */
  Gangle3    *angle,         /* (I) Angle of rotation, in radians
                                    (360 degrees = 2*pi radians).  Pass
                                    the value 0.0 to avoid rotating the
                                    segment. */
  Gscale3    *scale,         /* (I) X, Y, and Z scale factors.  Pass the
                                    value 1.0 to avoid scaling the
                                    segment. */
  Gcsw       coord,          /* (I) WC or NDC units switch (constant). */
  Gfloat     result[3][4]    /* (O) Output segment transformation matrix
                                    that results from the concatenation
                                    of the new scaling, rotation, and
                                    translation component values with
                                    the argument segtran. */
```

)

## Data Structures

```
typedef struct {        /* COORDINATE POINT */
  Gfloat     x;         /* X coordinate */
  Gfloat     y;         /* Y coordinate */
  Gfloat     z;         /* Z coordinate */
} Gpoint3;

typedef struct {        /* 3D ROTATION ANGLE STRUCTURE */
  Gfloat     x_angle;   /* rotation angle about x axis, in radians */
  Gfloat     y_angle;   /* rotation angle about y axis, in radians */
  Gfloat     z_angle;   /* rotation angle about z axis, in radians */
} Gangle3;

typedef struct {        /* 3D SCALING FACTORS */
  Gfloat     x_scale;   /* scaling factor about X axis */
  Gfloat     y_scale;   /* scaling factor about Y axis */
  Gfloat     z_scale;   /* scaling factor about Z axis */
} Gscale3;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcsw | GWC | The fixed point and shift vectors are WC values. |
| | GNDC | The fixed point and shift vectors are NDC values. |

## 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

gevaltran (

```
  Gpoint    *point,        /* (I) Fixed point for rotation, used as the
                                  only constant coordinate point during
                                  scaling and as the axis point during
                                  segment rotation.  */
  Gpoint    *shift,        /* (I) Shift vector.  Pass the value 0.0 to
                                  avoid translating the segment. */
  Gfloat    angle,         /* (I) Angle of rotation, in radians
                                  (360 degrees = 2*pi radians).  Pass
                                  the value 0.0 to avoid rotating the
                                  segment. */
  Gscale    *scale,        /* (I) X and Y scale factors.  Pass the value
                                  1.0 to avoid scaling the segment. */
  Gcsw      coord,         /* (I) WC or NDC unit switch. */
  Gfloat    result[2][3]   /* (O) Output segment transformation matrix that
                                  results from the concatenation of the
                                  new scaling, rotation, and translation
                                  component values.  You can use this value
                                  as an argument to SET SEGMENT
                                  TRANSFORMATION to establish a segment
                                  transformation. */
```

)

## Data Structures

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;

typedef struct {  /* SCALE VECTOR */
  Gfloat    x_scale;
  Gfloat    y_scale;
} Gscale;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcsw | GWC | The fixed point and shift vectors are WC values. |
| | GNDC | The fixed point and shift vectors are NDC values. |

## 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

gevaltran3 (

```
  Gpoint3   *point,        /* (I) Fixed point for rotation, used as the
                                  only constant coordinate point during
                                  scaling and as the axis point during
                                  segment rotation. */
  Gpoint3   *shift,        /* (I) Shift vector.  Pass the value 0 to
                                  avoid translating the segment. */
  Gangle3   *angle,        /* (I) Angle of rotation, in radians
                                  (360 degrees = 2*pi radians).  Pass
                                  the value 0 to avoid rotating the
                                  segment. */
  Gscale3   *scale,        /* (I) X, Y, and Z scale factors.  Pass the
                                  value 1.0 to avoid scaling the segment. */
  Gcsw      coord,         /* (I) WC or NDC unit switch (constant). */
  Gfloat    result[3][4]   /* (O) Output segment transformation matrix
                                  that results from the concatenation
                                  of the new scaling, rotation, and
                                  translation component values.  You can
                                  use this value as an argument to SET
                                  SEGMENT TRANSFORMATION 3 to establish
                                  a segment transformation. */
```

)

## Data Structures

```
typedef struct {        /* COORDINATE POINT */
  Gfloat    x;          /* X coordinate */
  Gfloat    y;          /* Y coordinate */
  Gfloat    z;          /* Z coordinate */
} Gpoint3;

typedef struct {        /* 3D ROTATION ANGLE STRUCTURE */
  Gfloat    x_angle;    /* rotation angle about X axis, in radians */
  Gfloat    y_angle;    /* rotation angle about Y axis, in radians */
  Gfloat    z_angle;    /* rotation angle about Z axis, in radians */
} Gangle3;

typedef struct {        /* 3D SCALING FACTORS */
  Gfloat    x_scale;    /* scaling factor about X axis */
  Gfloat    y_scale;    /* scaling factor about Y axis */
  Gfloat    z_scale;    /* scaling factor about Z axis */
} Gscale3;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcsw | GWC | The fixed point and shift vectors are WC values. |
| | GNDC | The fixed point and shift vectors are NDC values. |

## 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

gevalviewmaptran3 (

```
  Glimit    *window,       /* (I) VRC window limits */
  Glimit3   *viewport,     /* (I) NPC viewport limits */
  Gproj     proj_type,     /* (I) Projection type (constant) */
  Gpoint3   *prp,          /* (I) VRC projection reference point */
  Gfloat    vpd,           /* (I) VRC view plane distance */
  Gfloat    fpd,           /* (I) VRC front plane distance */
  Gfloat    bpd,           /* (I) VRC back plane distance */
  Gfloat    matrix[4] [4], /* (O) Returned orientation matrix */
  Gint      *error         /* (O) Error indicator */
```

)

### Data Structures

```
typedef struct {      /* COORDINATE LIMITS */
  Gfloat    xmin;     /* X minimum limit */
  Gfloat    xmax;     /* X maximum limit */
  Gfloat    ymin;     /* Y minimum limit */
  Gfloat    ymax;     /* Y maximum limit */
} Glimit;

typedef struct {      /* COORDINATE LIMITS */
  Gfloat    xmin;     /* X minimum limit */
  Gfloat    xmax;     /* X maximum limit */
  Gfloat    ymin;     /* Y minimum limit */
  Gfloat    ymax;     /* Y maximum limit */
  Gfloat    zmin;     /* Z minimum limit */
  Gfloat    zmax;     /* Z maximum limit */
} Glimit3;

typedef struct {      /* COORDINATE POINT */
  Gfloat    x;        /* X coordinate */
  Gfloat    y;        /* Y coordinate */
  Gfloat    z;        /* Z coordinate */
} Gpoint3;
```

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gproj | GPARALLEL | Parallel projection |
| | GPERSPECTIVE | 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

gevalvieworienttran3 (

```
  Gpoint3        *vrp,           /* (I) View reference point */
  Gpoint3        *vpn,           /* (I) View plane normal */
  Gpoint3        *vuv,           /* (I) View up vector */
  Gcsw           coord,          /* (I) WC or NDC switch (constant) */
  Gfloat         matrix[4] [4],  /* (O) View orientation matrix */
  Gint           *error          /* (O) Error indicator */
```

)

## Data Structures

```
typedef struct {    /* COORDINATE POINT */
  Gfloat    x;      /* X coordinate */
  Gfloat    y;      /* Y coordinate */
  Gfloat    z;      /* Z coordinate */
} Gpoint3;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcsw | GWC | The view reference point, view plane normal, and view up vector are WC values. |
| | GNDC | The view reference point, view plane normal, and view up vector are NDC values. |

## 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 ($4 \times 4$) 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

gselntran (

```
  Gint  transform   /* (I) 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

```
gsetclip (
  Gclip     indicator       /* (I) Clipping flag (constant) */
)
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gclip | GCLIP | Clipping enabled |
| | GNOCLIP | Clipping disabled |

## 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

gsetviewindex (

```
  Gint      index     /* Selected 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

gsetviewrep3 (

```
  Gint      ws,                 /* (I) Workstation identifier */
  Gint      index,              /* (I) View index number */
  Gfloat    orientation[4] [4], /* (I) Orientation matrix */
  Gfloat    mapping[4] [4],     /* (I) Mapping matrix */
  Glimit3   *clipping,          /* (I) Clipping limits in NPC points */
  Gclip     xy_clip,            /* (I) X-Y clipping indicator (constant) */
  Gclip     back_clip,          /* (I) Back clipping indicator (constant) */
  Gclip     front_clip          /* (I) Front clipping indicator (constant) */
```

)

### Data Structures

```
typedef struct {          /* COORDINATE LIMITS */
  Gfloat    xmin;         /* X minimum limit */
  Gfloat    xmax;         /* X maximum limit */
  Gfloat    ymin;         /* Y minimum limit */
  Gfloat    ymax;         /* Y maximum limit */
  Gfloat    zmin;         /* Z minimum limit */
  Gfloat    zmax;         /* Z maximum limit */
} Glimit3;
```

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gclip | GCLIP | Clipping enabled |
| | GNOCLIP | Clipping disabled |

### 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

gsetviewxformpr (

```
  Gint      ws,          /* (I) Workstation identifier */
  Gint      index,       /* (I) View index */
  Gint      reference,   /* (I) Reference for changing the view
                                transformation input priority */
  Gvpri     priority     /* (I) Priority flag (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gvpri | GHIGHER | Next higher priority |
| | GLOWER | 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

gsetviewport (

```
  Gint     transform,   /* (I) Transformation number.  Any subsequent calls to
                                output functions use the window and viewport
                                associated with this number. */
  Glimit   *viewport    /* (I) 2D viewport limits in NDC points.  Make
                                sure the X and Y values are located
                                within the default normalization viewport
                                boundaries. */
```

)

### Data Structures

```
typedef struct {    /* COORDINATE LIMITS */
  Gfloat   xmin;    /* X minimum limit */
  Gfloat   xmax;    /* X maximum limit */
  Gfloat   ymin;    /* Y minimum limit */
  Gfloat   ymax;    /* Y maximum limit */
} Glimit;
```

### 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

gsetviewport3 (

```
  Gint      transform,  /* (I) Transformation number. Any subsequent calls to
                               output functions use the window and viewport
                               associated with this number. */
  Glimit3   *viewport   /* (I) 3D viewport limits in NDC points.  Make
                               sure the X, Y, and Z values are located
                               within the default normalization
                               viewport boundaries. */
```

)

### Data Structures

```
typedef struct {          /* COORDINATE LIMITS */
  Gfloat      xmin;       /* X minimum limit */
  Gfloat      xmax;       /* X maximum limit */
  Gfloat      ymin;       /* Y minimum limit */
  Gfloat      ymax;       /* Y maximum limit */
  Gfloat      zmin;       /* Z minimum limit */
  Gfloat      zmax;       /* Z maximum limit */
} Glimit3;
```

### 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

gsetviewportinputpri (

```
  Gint     transform,    /* (I) Transformation whose priority is to
                                be set */
  Gint     reference,    /* (I) Reference for changing priority */
  Gvpri    priority      /* (I) Priority flag (constant) */
```

)

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gvpri | GHIGHER | 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

gsetwindow (

```
  Gint     transform,  /* (I) Transformation number.  Any subsequent calls
                              to output functions use the window and viewport
                              associated with this number. */
  Glimit   *window     /* (I) 2D window limits in WC points. */
```

)

## Data Structures

```
typedef struct {       /* COORDINATE LIMITS */
  Gfloat   xmin;       /* X minimum limit */
  Gfloat   xmax;       /* X maximum limit */
  Gfloat   ymin;       /* Y minimum limit */
  Gfloat   ymax;       /* Y maximum limit */
} Glimit;
```

## 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

gsetwindow3 (

```
  Gint       transform,    /* (I) Transformation number.  Any subsequent calls
                                 to output functions use the window and
                                 viewport associated with this number. */
  Glimit3    *window       /* (I) 3D window limits in WC points. */
```

)

### Data Structures

```
typedef struct {        /* COORDINATE LIMITS */
  Gfloat     xmin;      /* X minimum limit */
  Gfloat     xmax;      /* X maximum limit */
  Gfloat     ymin;      /* Y minimum limit */
  Gfloat     ymax;      /* Y maximum limit */
  Gfloat     zmin;      /* Z minimum limit */
  Gfloat     zmax;      /* Z maximum limit */
} Glimit3;
```

### 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

gsetwsviewport (

```
  Gint      ws,          /* (I) Workstation identifier */
  Glimit    *viewport    /* (I) Viewport limits in DC points */
```

)

## Data Structures

```
typedef struct {        /* COORDINATE LIMITS */
  Gfloat    xmin;       /* X minimum limit */
  Gfloat    xmax;       /* X maximum limit */
  Gfloat    ymin;       /* Y minimum limit */
  Gfloat    ymax;       /* Y maximum limit */
} Glimit;
```

## 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

gsetwsviewport3 (

```
  Gint     ws,          /* (I) Workstation identifier */
  Glimit3  *viewport    /* (I) Viewport limits in DC points */
```

)

## Data Structures

```
typedef struct {        /* COORDINATE LIMITS */
  Gfloat     xmin;      /* X minimum limit */
  Gfloat     xmax;      /* X maximum limit */
  Gfloat     ymin;      /* Y minimum limit */
  Gfloat     ymax;      /* Y maximum limit */
  Gfloat     zmin;      /* Z minimum limit */
  Gfloat     zmax;      /* Z maximum limit */
} Glimit3;
```

## 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

gsetwswindow (

```
  Gint      ws,          /* (I) Workstation identifier. */
  Glimit    *window      /* (I) Window limits in NDC points.  Make sure
                               these points are located within the
                               default workstation window boundary. */
```

)

## Data Structures

```
typedef struct {         /* COORDINATE LIMITS */
  Gfloat    xmin;        /* X minimum limit */
  Gfloat    xmax;        /* X maximum limit */
  Gfloat    ymin;        /* Y minimum limit */
  Gfloat    ymax;        /* Y maximum limit */
} Glimit;
```

## 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

gsetwswindow3 (

```
  Gint       ws,        /* (I) Workstation identifier. */
  Glimit3    *window    /* (I) Window limits in NDC points. Make sure
                              these points are located within the
                              default workstation window boundary. */
```

)

### Data Structures

```
typedef struct {        /* COORDINATE LIMITS */
  Gfloat     xmin;      /* X minimum limit */
  Gfloat     xmax;      /* X maximum limit */
  Gfloat     ymin;      /* Y minimum limit */
  Gfloat     ymax;      /* Y maximum limit */
  Gfloat     zmin;      /* Z minimum limit */
  Gfloat     zmax;      /* Z maximum limit */
} Glimit3;
```

### 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**

```
/*
 *  This program shows how using the ACCUMULATE TRANSFORMATION MATRIX
 *  function lets you add transformation components to a previously
 *  set transformation.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

#  include <stdio.h>
#  include <gks.h>

main ()
{
    Gconn         conn_id           = GWC_DEF;
    Gcsw          coord_switch      = GWC;
    Gevent        event;
    Gpoint        fixed_point;
    Gint          house             = 1;
    Gint          lower_left_corner = 1;
    Gint          num_pts           = 9;
    Gpoint        points[9];
    Gfloat        rotation          = 0;
    Gscale        scale;
    Gpoint        shift;
    Gfloat        time_out          = 5.00;
    Glimit        viewport;
    Gint          ws_id             = 1;
    Gwstype       ws_type           = GWS_DEF;
    Gfloat        xform_matrix[2][3];

/*  Open and activate GKS and the workstation environment. */

    gopengks (0, 0);
    gopenws (ws_id, &conn_id, &ws_type);
    gactivatews (ws_id);

/*
 *  Set the viewport limits for the specified normalization
 *  transformation.
 */

    viewport.xmin = viewport.ymin = 0.0;
    viewport.xmax = viewport.ymax = 0.5;

    gsetviewport (lower_left_corner, &viewport);

/*
 *  This call selects a normalization transformation with the
 *  new viewport.
 */

    gselntran (lower_left_corner);
    gsetclip (GNOCLIP);
```

(continued on next page)

**Example 7–1 (Cont.) Showing the Cumulative Effect of ACCUMULATE TRANSFORMATION MATRIX**

```
/*  Create the segment. */

    points[0].x  =  0.4;    points[0].y  = 0.1;
    points[1].x  =  0.1;    points[1].y  = 0.1;
    points[2].x  =  0.1;    points[2].y  = 0.7;
    points[3].x  =  0.4;    points[3].y  = 0.7;
    points[4].x  =  0.25;   points[4].y  = 0.9;
    points[5].x  =  0.1;    points[5].y  = 0.7;
    points[6].x  =  0.4;    points[6].y  = 0.1;
    points[7].x  =  0.4;    points[7].y  = 0.7;
    points[8].x  =  0.1;    points[8].y  = 0.1;

    gcreateseg (house);
    gpolyline (num_pts, points);
    gcloseseg ();

/*  Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (time_out, &event);

/*  Shift the house upwards and sidewards by 0.2 world coordinates. */

    shift.x = 0.2;          shift.y = 0.2;
    scale.x_scale = 1.0;    scale.y_scale = 1.0;
    fixed_point.x = 0.25;   fixed_point.y = 0.9;

    gevaltran (&fixed_point, &shift, rotation, &scale, coord_switch,
       xform_matrix);

/*
 *  Transform the segment and update the screen.  Calling SET SEGMENT
 *  TRANSFORMATION changes the segment transformation in the segment list,
 *  and sets flags in the workstation state list, telling GKS that the
 *  display surface is out of date and that an update is necessary.
 */

    gsetsegtran (house, xform_matrix);

/*
 *  Calling UPDATE WORKSTATION updates the position of the image on the
 *  workstation surface.  Wait 5 seconds.
 */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*
 *  Using ACCUMULATE TRANSFORMATION MATRIX, you can add transformation
 *  components to a previously set transformation.  The house gradually
 *  moves upward, one Y world coordinate point at a time.
 */

    gaccumtran (xform_matrix, &fixed_point, &shift, rotation, &scale,
       coord_switch, xform_matrix);

/*  Transform the segment and update the screen. */

    gsetsegtran (house, xform_matrix);

/*  Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);
```

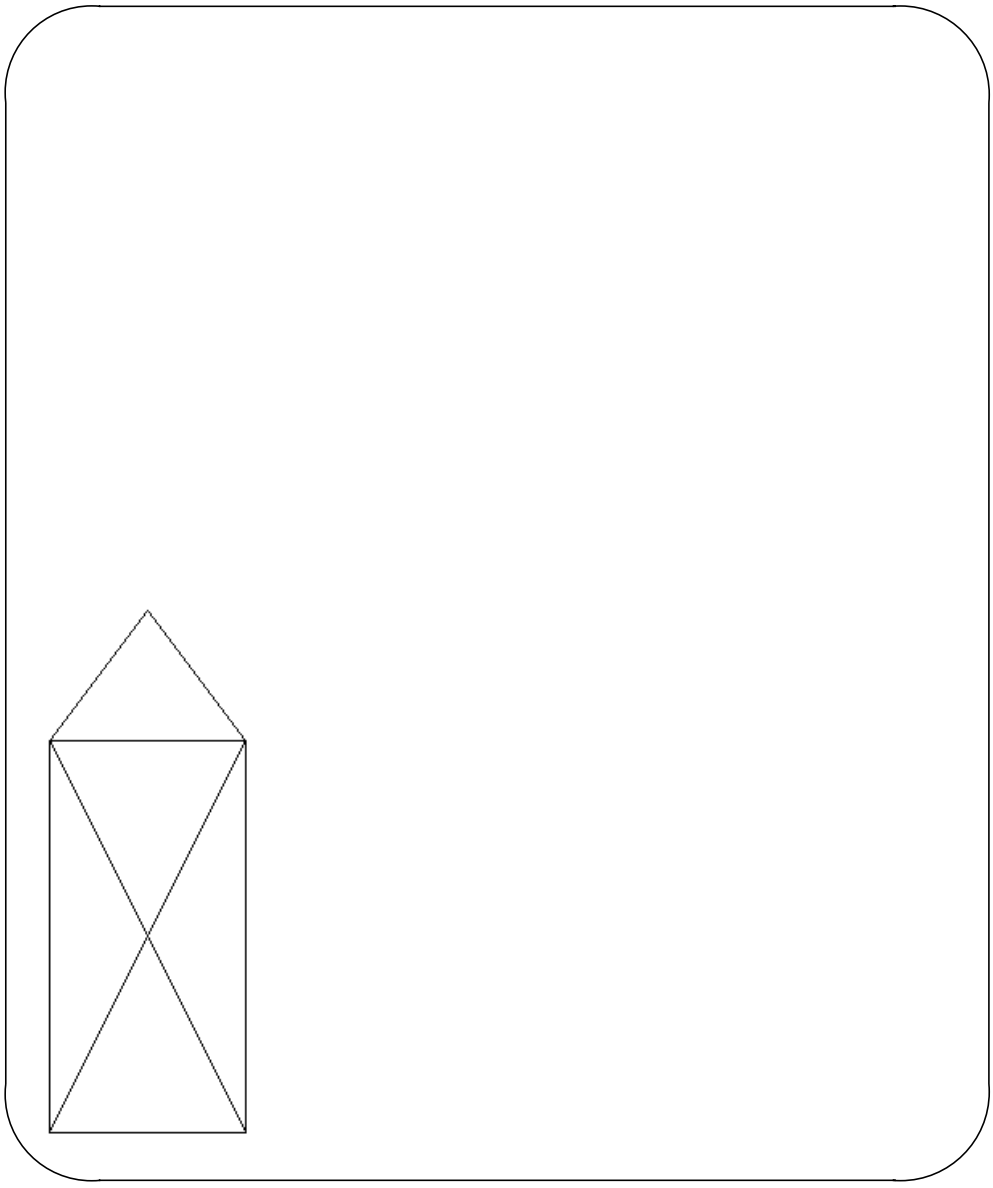
(continued on next page)

**Example 7–1 (Cont.) Showing the Cumulative Effect of ACCUMULATE TRANSFORMATION MATRIX**

```
/*  Again, shift the house upwards by 1 more world coordinate. */

    gaccumtran (xform_matrix, &fixed_point, &shift, rotation, &scale,
       coord_switch, xform_matrix);

/*  Transform the segment. */

    gsetsegtran (house, xform_matrix);

/*  Update the surface to initiate the change.  Wait 5 seconds. */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*  Deactivate and close the workstation environment and GKS. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();

}
```

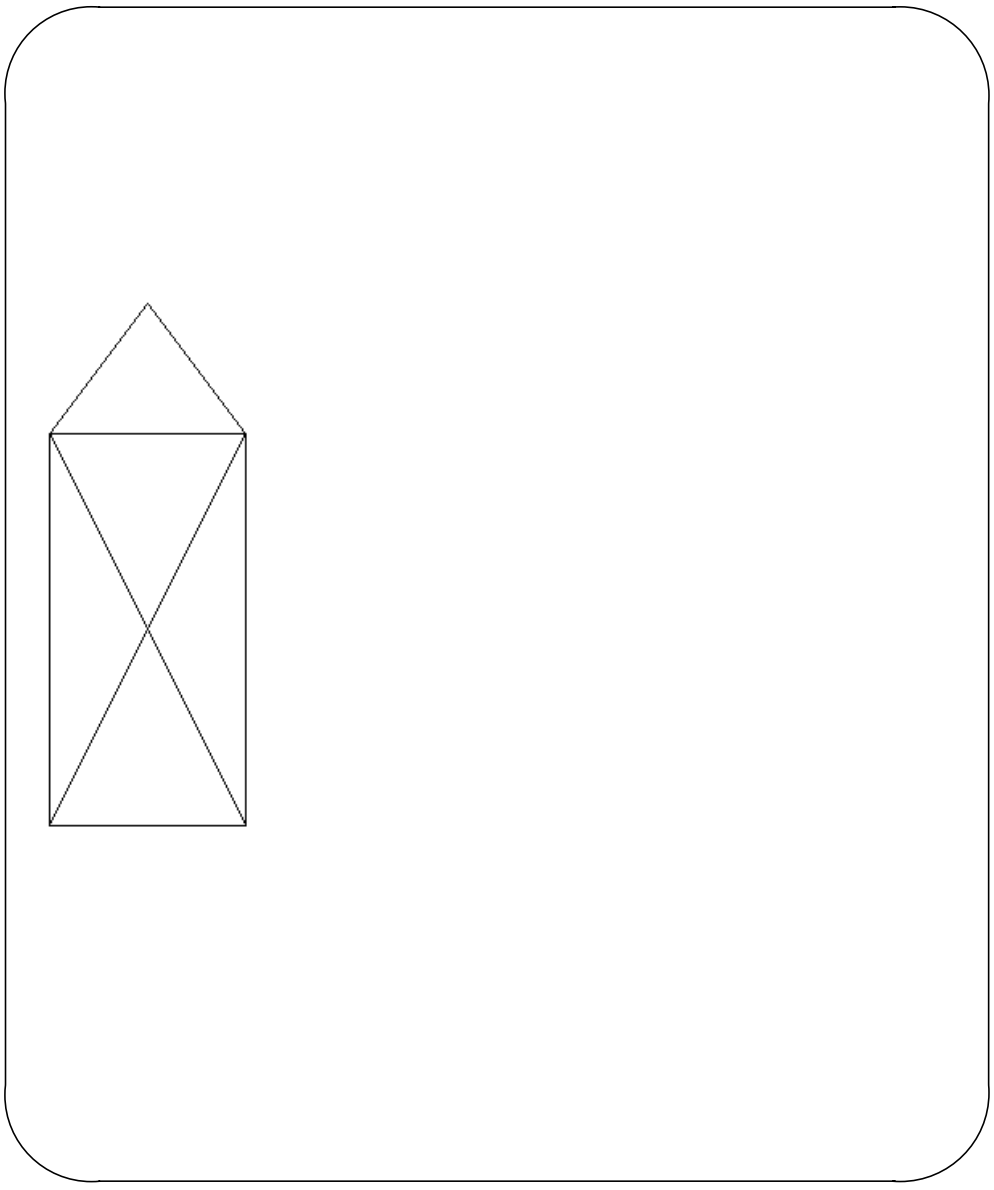
Figure 7–4 and Figure 7–5 show the first and last positions of the house. Each of the four house positions illustrates an added transformation component in the ACCUMULATE TRANSFORMATION MATRIX function.

**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

```
/*
 *  This program transforms the house contained in a segment.  The
 *  program shows the effects of segment transformation through the
 *  use of the EVALUATE TRANSFORMATION MATRIX function.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

#  include <stdio.h>
#  include <gks.h>           /* C binding definition file */

main ()
{
    Gconn         conn_id       = GWC_DEF;
    Gcsw          coord_switch  = GWC;
    Gevent        event;
    Gpoint        fixed_point;
    Gint          house         = 1;
    Gint          lower_left_corner = 1;
    Gint          num_pts       = 9;
    Gpoint        points[9];
    Gfloat        rotation;
    Gscale        scale;
    Gpoint        shift;
    Gfloat        time_out      = 5.00;
    Glimit        viewport;
    Gint          ws_id         = 1;
    Gwstype       ws_type       = GWS_DEF;
    Gfloat        xform_matrix[2][3];

/*  Open and activate GKS and the workstation environment. */

    gopengks (0, 0);
    gopenws (ws_id, &conn_id, &ws_type);
    gactivatews (ws_id);

/*  Set the viewport limits for the specified normalization transformation. */

    viewport.xmin = viewport.ymin = 0.0;
    viewport.xmax = viewport.ymax = 0.5;

    gsetviewport (lower_left_corner, &viewport);

/*
 *  This call selects a normalization transformation with the
 *  new viewport.
 */

    gselntran (lower_left_corner);
    gsetclip (GNOCLIP);

/*  Create the segment. */

    points[0].x  =  0.4;    points[0].y  = 0.1;
    points[1].x  =  0.1;    points[1].y  = 0.1;
    points[2].x  =  0.1;    points[2].y  = 0.7;
    points[3].x  =  0.4;    points[3].y  = 0.7;
    points[4].x  =  0.25;   points[4].y  = 0.9;
    points[5].x  =  0.1;    points[5].y  = 0.7;
    points[6].x  =  0.4;    points[6].y  = 0.1;
    points[7].x  =  0.4;    points[7].y  = 0.7;
    points[8].x  =  0.1;    points[8].y  = 0.1;
```

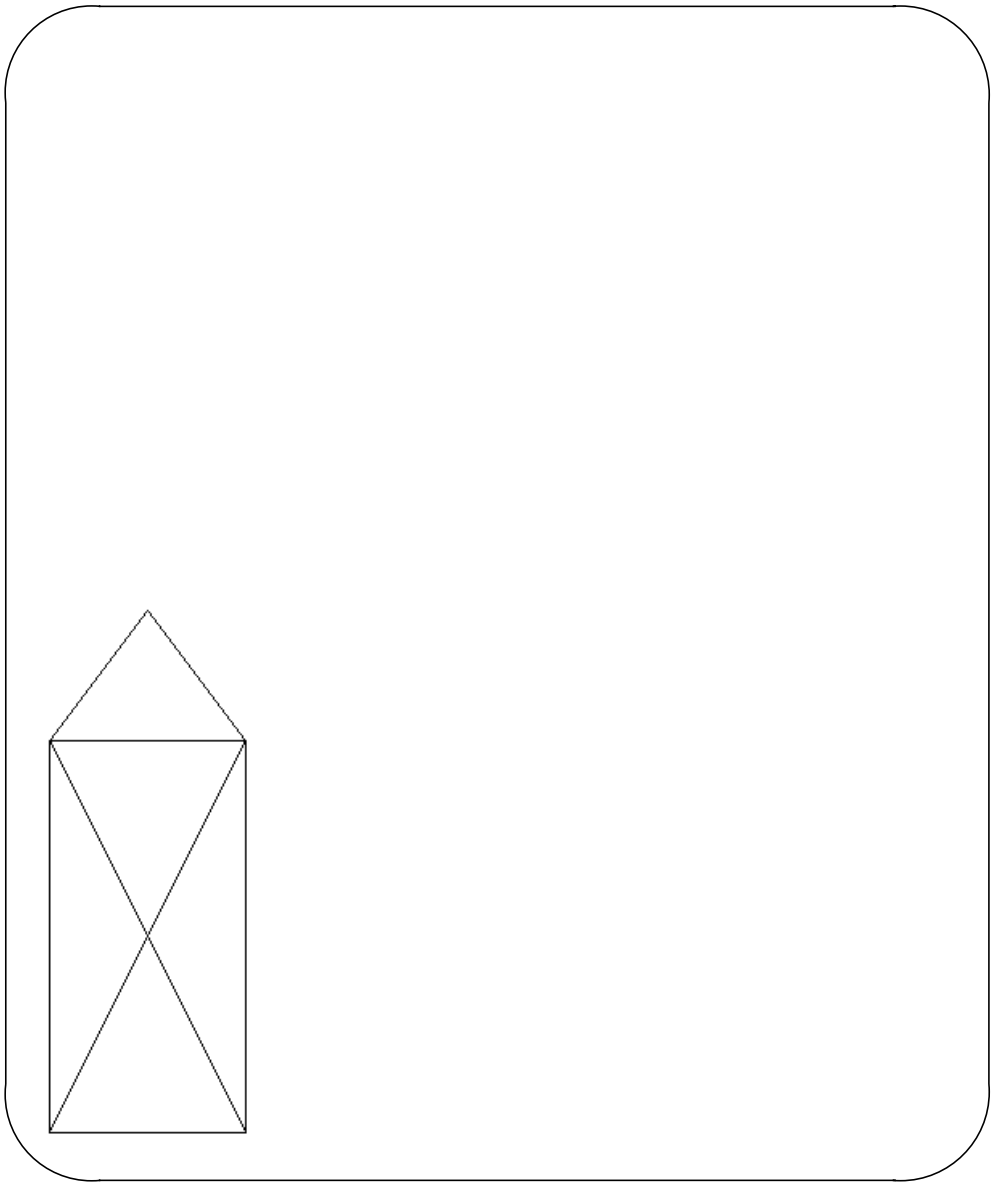
(continued on next page)

**Example 7–2 (Cont.) The Effects of a Segment Transformation**

```
    gcreateseg (house);
    gpolyline (num_pts, points);
    gcloseseg ();

/*  Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (time_out, &event);

/*  Rotation equals pi divided by 6 (30 degrees). */

    rotation = 3.14/6.0;

/*
 *  You can change the segment transformation that affects the
 *  rotation, scaling, and translation components of segment
 *  appearance.  The EVALUATE TRANSFORMATION MATRIX call assists in the
 *  creation of a new transformation matrix, that will permit you to
 *  specify rotation, scaling, and translation values.
 */

    shift.x = 0.0;          shift.y = 0.2;
    scale.x_scale = 0.5;    scale.y_scale = 0.5;
    fixed_point.x = 0.25;   fixed_point.y = 0.9;

    gevaltran (&fixed_point, &shift, rotation, &scale, coord_switch,
       xform_matrix);

/*  Transform the segment. */

    gsetsegtran (house, xform_matrix);

/*
 *  The UPDATE WORKSTATION function updates the position of the image on
 *  the workstation surface.  All output not contained in segments is lost.
 *  Wait 5 seconds.
 */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*  Deactivate and close the workstation environment and GKS. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();

}
```

Figure 7–6 shows the house before the segment transformation.

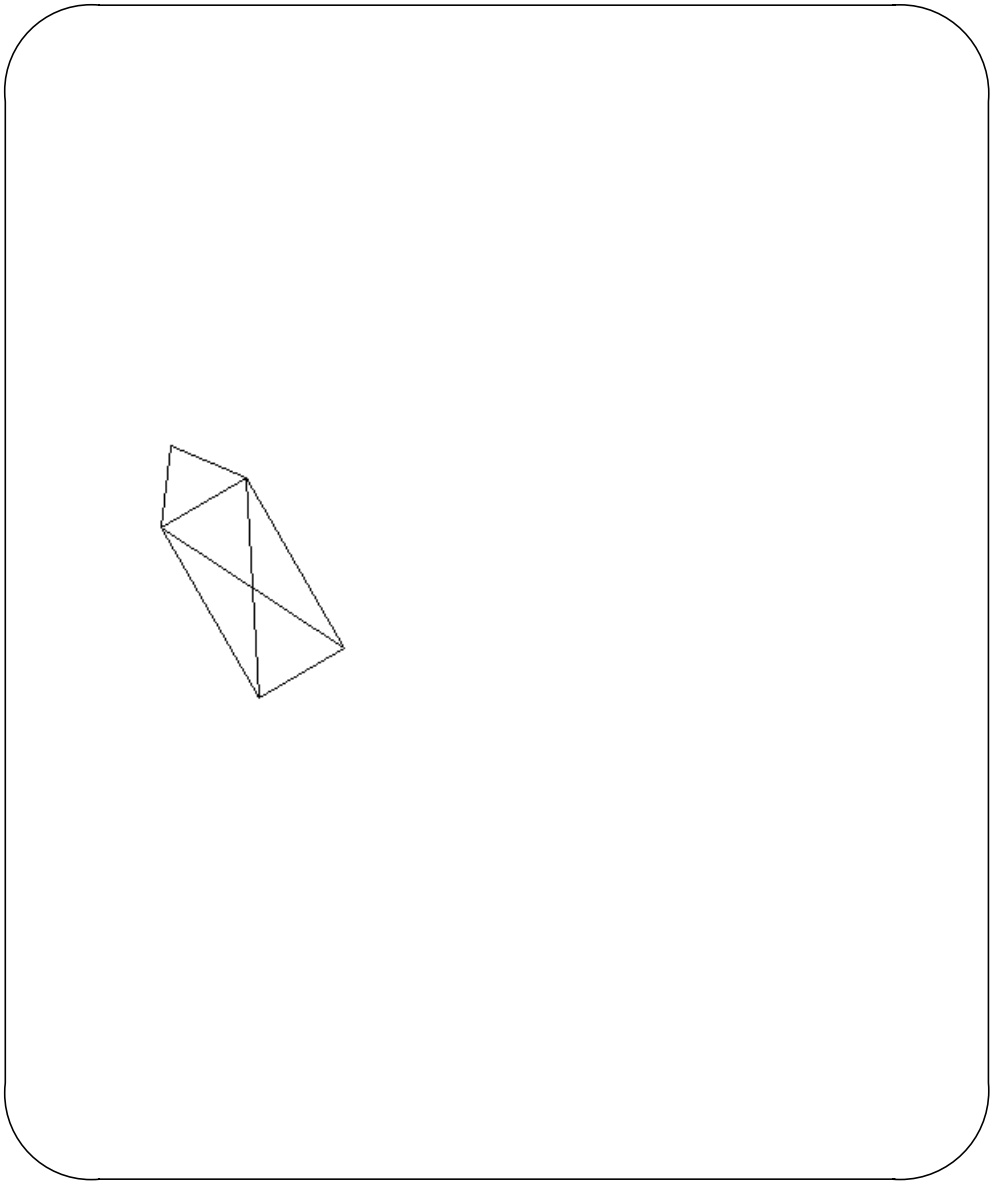
**Figure 7–6 Output Prior to Segment Transformation**



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

```
/*
 *  This program illustrates the SET CLIPPING INDICATOR function.  It
 *  generates a "tall, thin house" that overlaps the normalization window
 *  and viewport.  You can see the overlapping portion if clipping is
 *  disabled.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

#  include <stdio.h>
#  include <gks.h>

main ()
{
    Gconn         conn_id           = GWC_DEF;
    Gevent        event;
    Gint          half              = 1;
    Gint          low_left_corner   = 1;
    Gint          num_pts           = 9;
    Gpoint        points[9];
    Gpoint        points_2[5];
    Gfloat        time_out          = 5.00;
    Glimit        viewport;
    Glimit        window;
    Gint          ws_id             = 1;
    Gwstype       ws_type           = GWS_DEF;

/*  Open and activate GKS and the workstation environment. */

    gopengks (0, 0);
    gopenws (ws_id, 0, &ws_type);
    gactivatews (ws_id);

/*
 *  Outlining the default world window results in the outlining of
 *  the NDC plane, the workstation window, and the workstation
 *  viewport, by default.
 */

    points[0].x  =  0.0;    points[0].y  = 0.0;
    points[1].x  =  0.5;    points[1].y  = 0.5;
    points[2].x  =  0.0;    points[2].y  = 0.5;
    points[3].x  =  0.0;    points[3].y  = 0.0;

    gpolyline (5, points_2);

/*
 *  Set the normalization window to be half of the default
 *  plane and assign the normalization transformation low_left_corner.
 *  The house is cut in half by the window boundary.  Set the viewport to be
 *  the lower left corner of the NDC space and assign this normalization
 *  transformation the value 1.
 */

    viewport.xmin = viewport.ymin = 0.0;
    viewport.xmax = viewport.ymax = 0.5;
    window.xmin = window.ymin = 0.0;
    window.xmax = 0.9;
    window.ymax = 0.5;

    gsetwindow (half, &window);
    gsetviewport (low_left_corner, &viewport);
```

(continued on next page)

**Example 7–3 (Cont.) Controlling Clipping at the World Viewport**

```
/*
 *  Select the normalization transformation number 1 which has a
 *  smaller window by half and a viewport that is the lower left
 *  corner of the NDC space.  By default, clipping is enabled; you
 *  only see half the house.
 */

    gselntran (low_left_corner);

    points[0].x  =  0.4;    points[0].y  = 0.1;
    points[1].x  =  0.1;    points[1].y  = 0.1;
    points[2].x  =  0.1;    points[2].y  = 0.7;
    points[3].x  =  0.4;    points[3].y  = 0.7;
    points[4].x  =  0.25;   points[4].y  = 0.9;
    points[5].x  =  0.1;    points[5].y  = 0.7;
    points[6].x  =  0.4;    points[6].y  = 0.1;
    points[7].x  =  0.4;    points[7].y  = 0.7;
    points[8].x  =  0.1;    points[8].y  = 0.1;

    gpolyline (num_pts, points);

/*  Release deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*
 *  Once you disable clipping and redraw the picture, GKS maps the
 *  entire house to NDC space and eventually the workstation surface.
 */

    gsetclip (GNOCLIP);

/*
 *  Draw the same house, using the same windows and viewports, but
 *  with clipping disabled.  Now you can see the portion of the house
 *  that overlaps the window and viewport.
 */

    gpolyline (num_pts, points);

/* Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*  Deactivate and close the workstation environment and GKS. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();

}
```

Figure 7–8 illustrates the house while clipping is enabled.

**Figure 7–8 SET CLIPPING INDICATOR with Clipping Enabled**

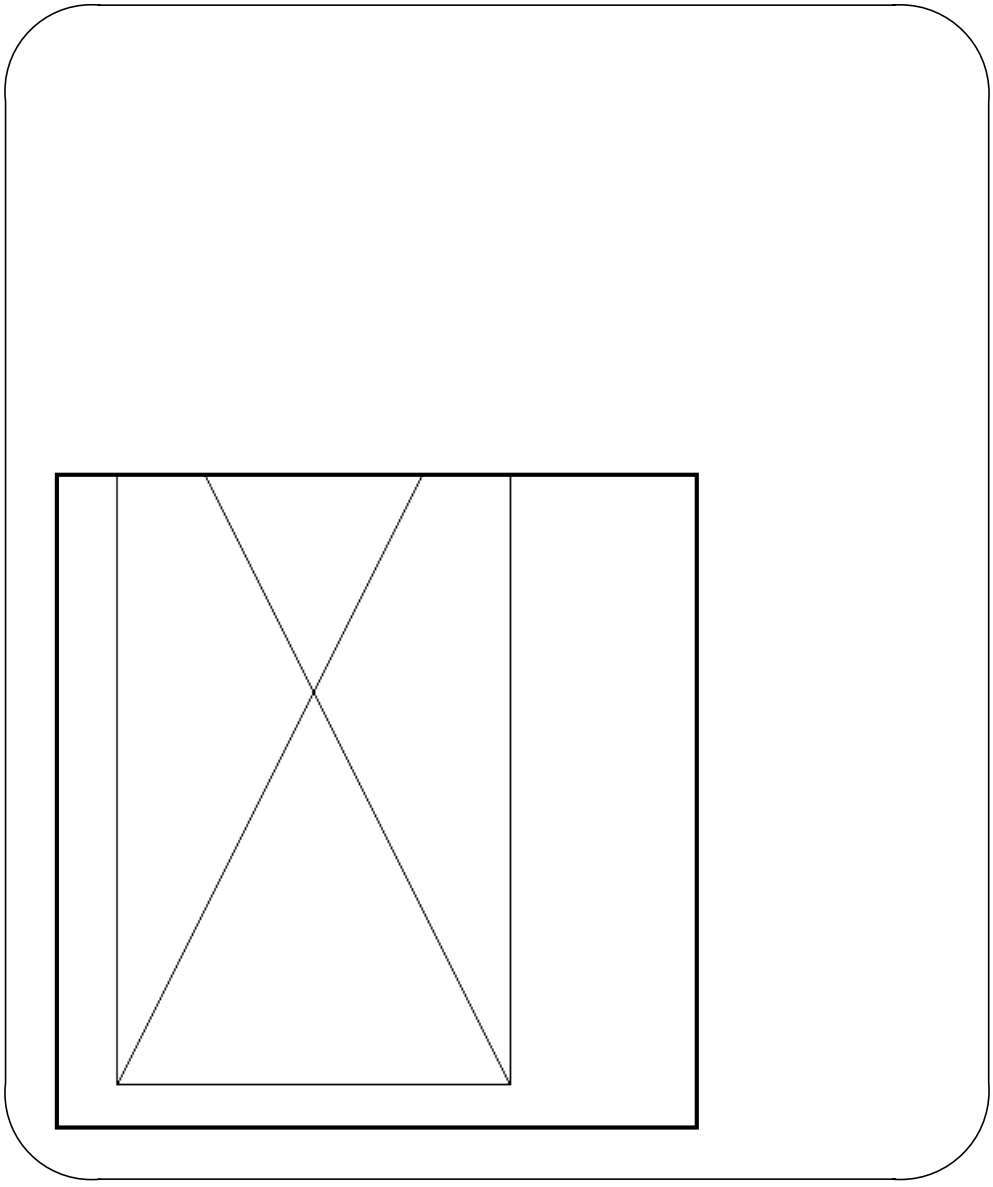
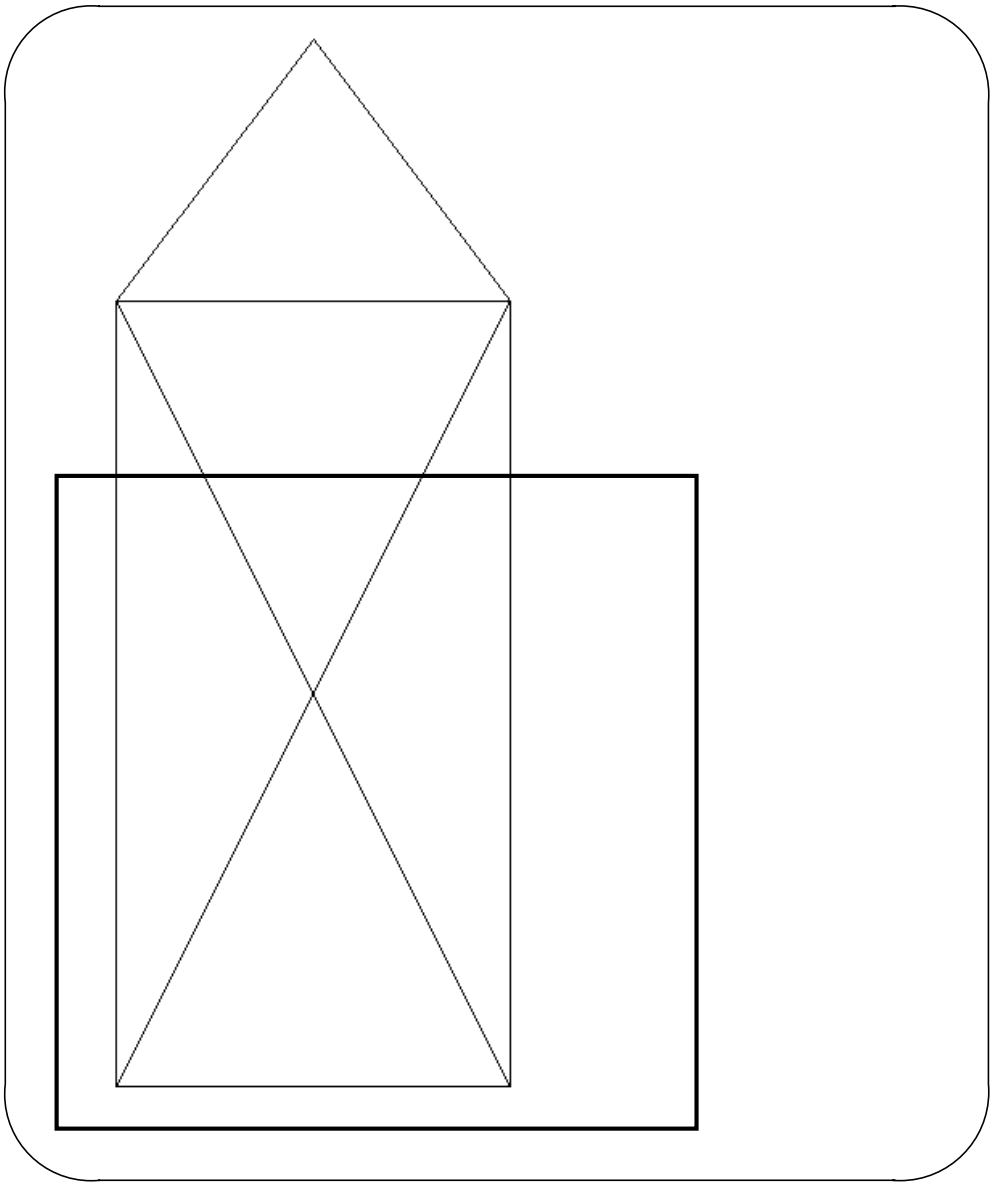



Figure 7–9 illustrates the house while clipping is disabled. The house overlaps the normalization window and viewport.

**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

```
/*
 *  This program illustrates the SET WORKSTATION VIEWPORT function.
 *  This program uses the default normalization transformations,
 *  generates a "tall, thin house," updates the screen,
 *  changes the workstation viewport to the lower left
 *  corner of the display surface, and generates the output again.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

#  include <stdio.h>
#  include <gks.h>      /* C binding definition file */

main ()
{

    Gdspsize      dspsz;
    Gmodws        dyn;
    Gint          error_status;
    Gevent        event;
    Gint   house  = 1;
    Gfloat        max_x;
    Gfloat        max_y;
    Gint          num_pts = 9;
    Gpoint   points[9];
    Gpoint   outline[5];
    Gfloat        time_out = 5.00;
    Glimit        viewport ;
    Gint          ws_id  = 1;
    Gwstype       ws_type = GWS_DEF;

/*  Open GKS and the workstation environment. */

    gopengks (0, 0);
    gopenws (ws_id, 0, &ws_type);
    gactivatews (ws_id);

/*
 *  Outlining the default window results in outlining
 *  the NDC plane, the workstation window, and the workstation
 *  viewport, by default.
 */

    outline[0].x =  0.0;    outline[0].y = 0.0;
    outline[1].x =  1.0;    outline[1].y = 0.0;
    outline[2].x =  1.0;    outline[2].y = 1.0;
    outline[3].x =  0.0;    outline[3].y = 1.0;
    outline[4].x =  0.0;    outline[4].y = 0.0;

    gpolyline (5, outline);
```

(continued on next page)

**Example 7–4 (Cont.) Establishing a Workstation Viewport**

```
/*
 *  This code assumes the default normalization transformation.
 *  GKS maps the default window to the default viewport, then
 *  maps the default workstation window to the default viewport.
 */

    points[0].x  =  0.4;    points[0].y  = 0.1;
    points[1].x  =  0.1;    points[1].y  = 0.1;
    points[2].x  =  0.1;    points[2].y  = 0.7;
    points[3].x  =  0.4;    points[3].y  = 0.7;
    points[4].x  =  0.25;   points[4].y  = 0.9;
    points[5].x  =  0.1;    points[5].y  = 0.7;
    points[6].x  =  0.4;    points[6].y  = 0.1;
    points[7].x  =  0.4;    points[7].y  = 0.7;
    points[8].x  =  0.1;    points[8].y  = 0.1;

    gcreateseg (house);
    gpolyline (num_pts, points);
    gcloseseg ();

/*  Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*
 *  The function INQUIRE DISPLAY SPACE SIZE returns the maximum X
 *  and Y display surface coordinates in the arguments device_max_x
 *  and device_max_y.
 */

    ginqdisplaysize (&ws_type, &dspsz, &error_status);

/*
 *  The function SET WORKSTATION VIEWPORT changes the workstation
 *  viewport to the lower left corner of the screen.  The picture
 *  being displayed is still the same (the default workstation
 *  window), but the space on the workstation surface used to
 *  display the same picture has changed.
 */

    viewport.xmin = 0.0;
    viewport.xmax = dspsz.device.x/2.0;
    viewport.ymin = 0.0;
    viewport.ymax = dspsz.device.y/2.0;

    gsetwsviewport (ws_id, &viewport);

/*
 *  Update the screen so the workstation can use the new workstation viewport
 *  (as noted in the function description section).
 */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);
```

(continued on next page)

**Example 7–4 (Cont.) Establishing a Workstation Viewport**

```
/*
 *  Check whether the workstation viewport change required an implicit
 *  regeneration (IRG), thereby deleting all information not retained
 *  in a segment.
 */

    ginqmodwsattr (&ws_type, &dyn, &error_status);
    if (dyn.wstran == GIRG)
       {
       outline[0].x =  0.0;    outline[0].y = 0.0;
       outline[1].x =  1.0;    outline[1].y = 0.0;
       outline[2].x =  1.0;    outline[2].y = 1.0;
       outline[3].x =  0.0;    outline[3].y = 1.0;
       outline[4].x =  0.0;    outline[4].y = 0.0;

       gpolyline (5, outline);
       }

/*  Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*  Close the workstation environment and GKS. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();

}
```

Figure 7–10 illustrates how the house is displayed using the default normalization transformation.

**Figure 7–10 Output Using the Default Normalization Transformation**

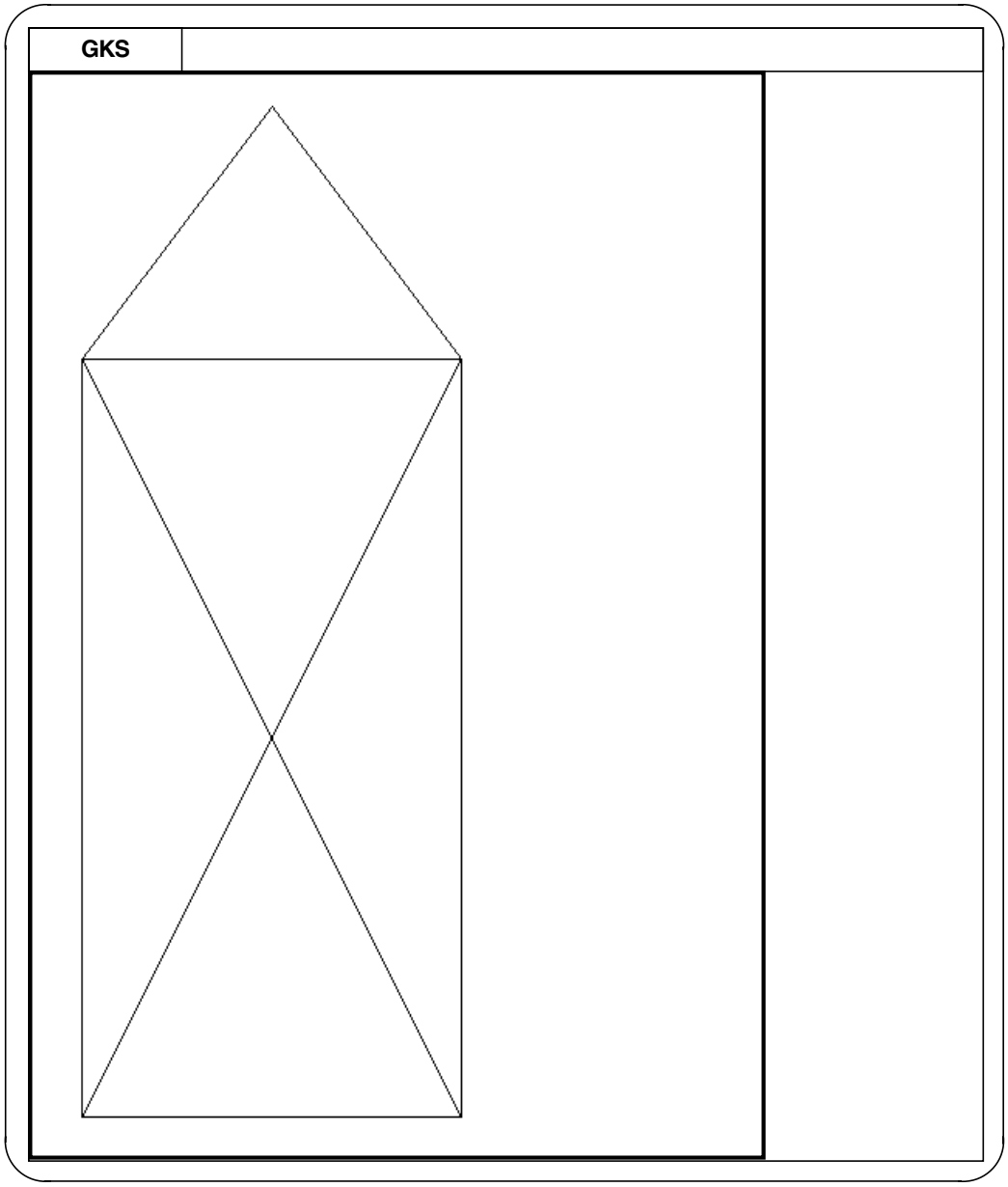
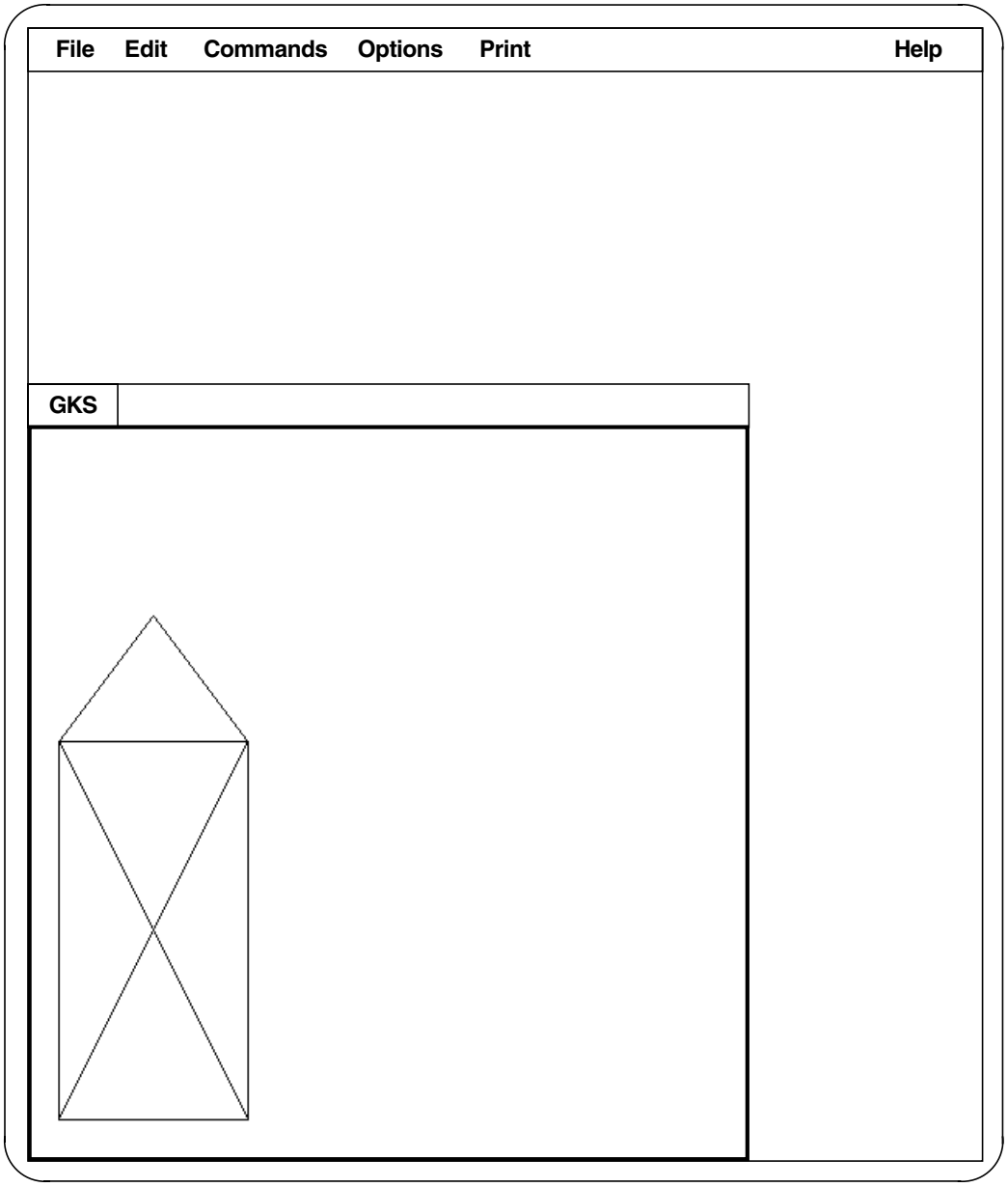



Figure 7–11 illustrates how the house is displayed with changes to the workstation viewport.

**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 GBUNDLED 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 GWS_WISS (value 5) as the workstation type. When you open WISS, you can specify GWC_DEF as the connection identifier argument. (If you specify GWS_WISS, 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 GPERFORM 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 GPERFORM.

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.<br>• ASSOCIATE SEGMENT WITH WORKSTATION<br>• DELETE SEGMENT<br>• DELETE SEGMENT FROM WORKSTATION<br>• SET SEGMENT PRIORITY<br>• SET SEGMENT TRANSFORMATION<br>• SET VISIBILITY<br>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 (GUNDETECTABLE).

To pick a segment, it must be both detectable and visible (GVISIBLE). 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 (GNORMAL).

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 the NDC plane and then performs the rotation or scaling.

If you want to transform the segment as stored on the NDC space (regardless of the current normalization transformation), specify an NDC point as your fixed point.

Next, if you want to scale or to rotate the segment, you must decide which point in the segment to use as 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. DEC GKS rotates the segment on the axis of the fixed point by the radian specified as the angle of rotation. Positive rotation values rotate counter clockwise; negative rotation values rotate clockwise. If you do not wish to rotate the segment, you can specify 0.0 for the angle of rotation.

Finally, if you decide to scale the segment, you need to establish the **scale factors**. You express a scale factor as two real number values; DEC GKS multiplies the X and Y segment coordinate values by the scale factor components to determine the new size of the segment. If you do not want to scale the segment (keeping the segment the same size), specify the value 1.0 for 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 and ACCUMULATE TRANSFORMATION MATRIX. The function EVALUATE TRANSFORMATION MATRIX has the following function syntax:

```
gevaltran ( point, shift, angle, scale, coord, result ) ;
```

After evaluating the first five arguments, EVALUATE TRANSFORMATION MATRIX establishes the appropriate transformation matrix and writes the 6-element array of real numbers to the last argument *result*. 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:

```
gaccumtran ( segtran, point, shift, angle, scale, coord, result ) ;
```

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, 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 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 (GVISIBLE).

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 (GVISIBLE). 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
INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES
INQUIRE LEVEL OF GKS
INQUIRE NAME OF OPEN SEGMENT
INQUIRE OPERATING STATE VALUE
INQUIRE PICK DEVICE STATE
INQUIRE SEGMENT ATTRIBUTES
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

gassocsegws (

```
  Gint      ws,       /* (I) Workstation identifier. */
  Gint      seg       /* (I) Segment name that identifies a segment
                           that is 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

gcloseseg ( )

### 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

gcopysegws (

```
  Gint      ws,       /* (I) Workstation identifier.  The workstation
                             cannot be of type GWISS. */
  Gint      seg       /* (I) 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 stored with the primitive, sent through the viewing pipeline, and are finally transformed by the workstation transformation and output.

Primitives are clipped by the clipping volume 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

gcreateseg (

```
  Gint       seg       /* (I) Segment name.  You cannot use the same
                              identifier to name 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

gdelseg (

```
  Gint    seg     /* (I) 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

gdelsegws (

```
  Gint      ws,        /* (I) Workstation identifier */
  Gint      seg        /* (I) 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

ginsertseg (

```
  Gint    seg,                  /* (I) Segment name */
  Gfloat  insert_xform[2][3]    /* (I) 2D transformation matrix */
```

)

## 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 the following attributes in the specified segment:

- All clipping indicators and clipping volumes
- All view indexes
- All hidden line and hidden surface removal (HLHSR) identifiers

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

ginsertseg3 (

```
  Gint     seg,                    /* (I) Segment name */
  Gfloat   insert_xform[3] [4]     /* (I) 3D transformation matrix */
```

)

### 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 the following attributes in the specified segment:

- All clipping indicators and clipping volumes
- All view indexes
- All hidden line and hidden surface removal (HLHSR) identifiers

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

grenameseg (

```
  Gint      old,      /* (I) Old segment name */
  Gint      new       /* (I) 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

gsetdet (

```
  Gint      seg,              /* (I) Segment name */
  Gsegdet   detectability     /* (I) Detectability flag */
```

)

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gsegdet | GUNDETECTABLE | Segment cannot be picked. This is the default value. |
| | GDETECTABLE | 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

gsethighlight (

```
  Gint       seg,              /* (I) Segment name */
  Gseghi     highlighting      /* (I) Highlighting flag (constant) */
```

)

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gseghi | GNORMAL | DEC GKS does not highlight the segment. This is the default value. |
| | GHIGHLIGHTED | 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

gsetsegpri (

```
  Gint       seg,          /* (I) Segment name. */
  Gfloat     priority      /* (I) Priority 0.0 -> 1.0.  The default
                                  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

gsetsegtran (

```
  Gint      seg,              /* (I) Segment name. */
  Gfloat    transform[2][3]   /* (I) 2D transformation matrix created
                                   previously by a call to either
                                   EVALUATE TRANSFORMATION MATRIX or
                                   ACCUMULATE TRANSFORMATION MATRIX. */
```

)

### 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

gsetsegtran3 (

```
Gint       seg,                /* (I) Segment name. */
Gfloat     transform[3] [4]    /* (I) 3D transformation matrix created
                                      previously by a call to either
                                      EVALUATE TRANSFORMATION MATRIX 3
                                      or ACCUMULATE TRANSFORMATION
                                      MATRIX 3. */
```

)

### 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

gsetvis (

```
  Gint        seg,            /* (I) Segment name */
  Gsegvis     visibility      /* (I) Visibility flag */
```

)

### Constants

| Data Type | Constant | Description |
|---|---|---|
| Gsegvis | GVISIBLE | DEC GKS shows the segment on the workstation surface. This is the default value. |
| | GINVISIBLE | DEC GKS does not show the segment on the workstation surface. |

### 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**

```
/*
 *  This program draws a house in the lower left corner of the
 *  screen and a line of text in the upper left corner. The program
 *  redraws the segments to show the ASSOCIATE SEGMENT WITH
 *  WORKSTATION and COPY SEGMENT TO WORKSTATION functions.
 *
 *  NOTE: To keep the example concise. no error checking is performed.
 */

#  include <stdio.h>
#  include <gks.h>

#define NUM_PTS 9

main ()
{
    Gevent        event;
    Gint          house             = 1;
    Gfloat        larger            = .03;
    Gint          lower_left_corner = 1;
    Gint          num_pts           = NUM_PTS;
    Gpoint        points[NUM_PTS];
    Gchar         *text             = "Associated segment.";
    Gfloat        time_out          = 5.00;
    Gint          title             = 2;
    Gint          upper_left_corner = 2;
    Glimit        viewport;
    Glimit        viewport2;
    Gint          wiss              = 2;
    Gpoint        world_pt;
    Gint          ws_id             = 1;
    Gwstype       ws_type           = GWS_DEF;
    Gwstype       ws_type_wiss      = GWS_WISS;

/*  Open GKS and the workstation environments. */

    gopengks (0, 0);
    gopenws (ws_id, 0, &ws_type);
    gopenws (wiss, 0, &ws_type_wiss);

/*
 *  By activating only the WISS workstation, GKS stores created
 *  segments only on the WISS workstation.  The screen remains clear.
 */

    gactivatews (wiss);
```

(continued on next page)

**Example 8–1 (Cont.) Comparing ASSOCIATE SEGMENT WITH WORKSTATION and COPY SEGMENT TO WORKSTATION**

```
/*
 *  Set the viewport limits for the specified normalization
 *  transformations.
 */

    viewport.xmin = viewport.ymin= 0.0;
    viewport.xmax = viewport.ymax= 0.5;

    gsetviewport (lower_left_corner, &viewport);

    viewport2.xmin = 0.0;
    viewport2.xmax = 0.5;
    viewport2.ymin = 0.5;
    viewport2.ymax = 1.0;

    gsetviewport (upper_left_corner, &viewport2);

/*  Create two segments and store them on the WISS workstation. */

    gcreateseg (house);
    gselntran (lower_left_corner);

    points[0].x = 0.4;    points[0].y = 0.1;
    points[1].x = 0.1;    points[1].y = 0.1;
    points[2].x = 0.1;    points[2].y = 0.7;
    points[3].x = 0.4;    points[3].y = 0.7;
    points[4].x = 0.25;   points[4].y = 0.9;
    points[5].x = 0.1;    points[5].y = 0.7;
    points[6].x = 0.4;    points[6].y = 0.1;
    points[7].x = 0.4;    points[7].y = 0.7;
    points[8].x = 0.1;    points[8].y = 0.1;

    gpolyline (num_pts, points);
    gcloseseg ();

    gcreateseg (title);
    gselntran (upper_left_corner);
    gsetcharheight (larger);

    world_pt.x = 0.1; world_pt.y = 0.5;

    gtext (&world_pt, text);
    gcloseseg ();

/*  Activate the ws_id workstation. */

    gactivatews (ws_id);

/*
 *  By associating the segment containing text, the workstation stores
 *  the segment. By copying the segment containing the house, GKS draws
 *  the primitives to the display screen, but the workstation does not
 *  store the segment.
 */

    gassocsegws (ws_id, title);
    gcopysegws (ws_id, house);

/*  Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (time_out, &event);
```

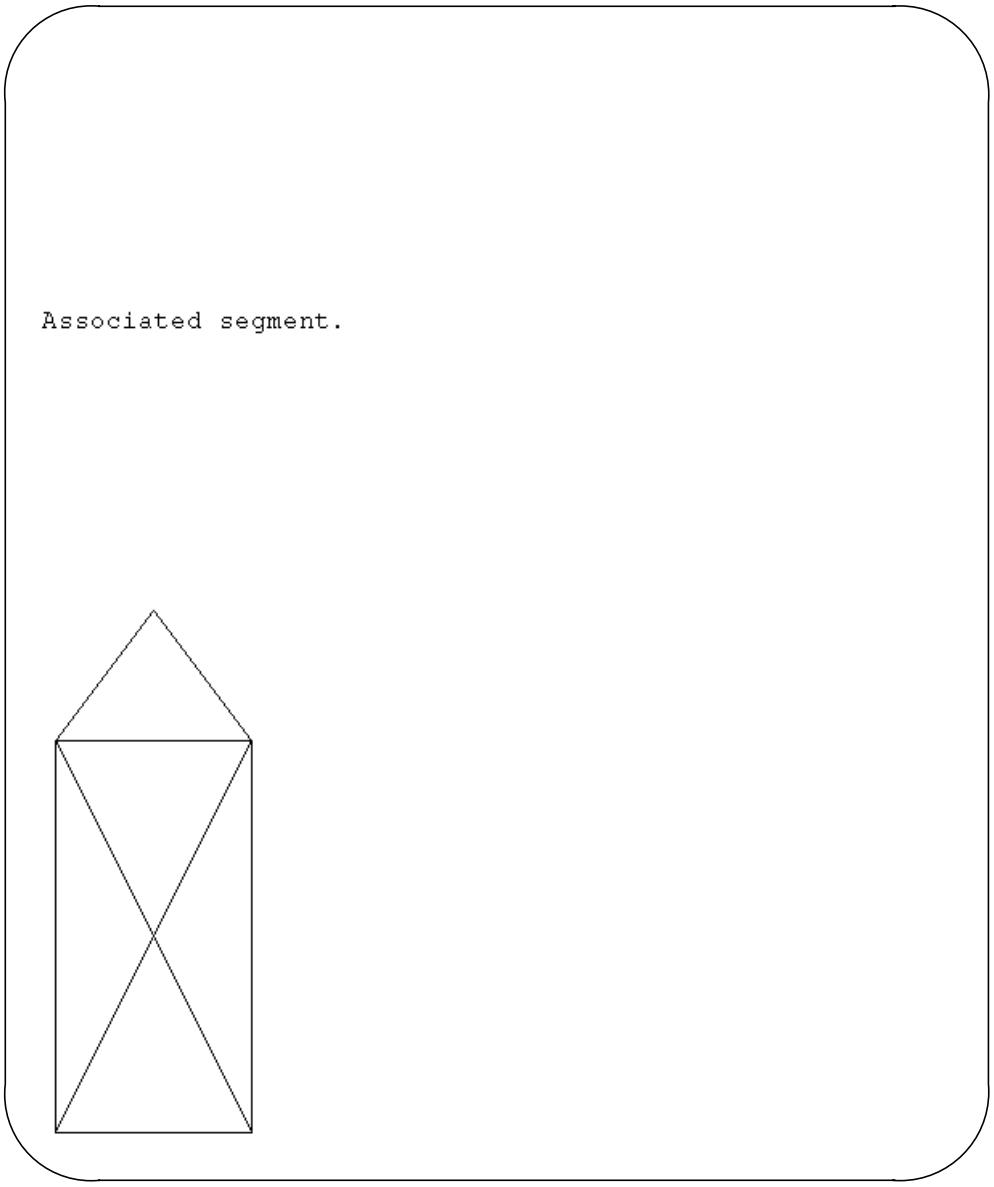
(continued on next page)

**Example 8–1 (Cont.) Comparing ASSOCIATE SEGMENT WITH WORKSTATION and COPY SEGMENT TO WORKSTATION**

```
/*
 *  When you redraw the segments, you force an update to the screen
 *  and eliminate primitives not contained in segments.  The house
 *  disappears and the text remains on the display screen because it
 *  was stored as a segment.
 */

    gredrawsegws (ws_id);

/*  Only the text is displayed.  Wait 5 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (time_out, &event);

/*  Deactivate and close the workstation environments and GKS. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gdeactivatews (wiss);
    gclosews (wiss);
    gclosegks ();
}
```

Figure 8–1 shows the two segments (the house and the line of text) on the display surface.

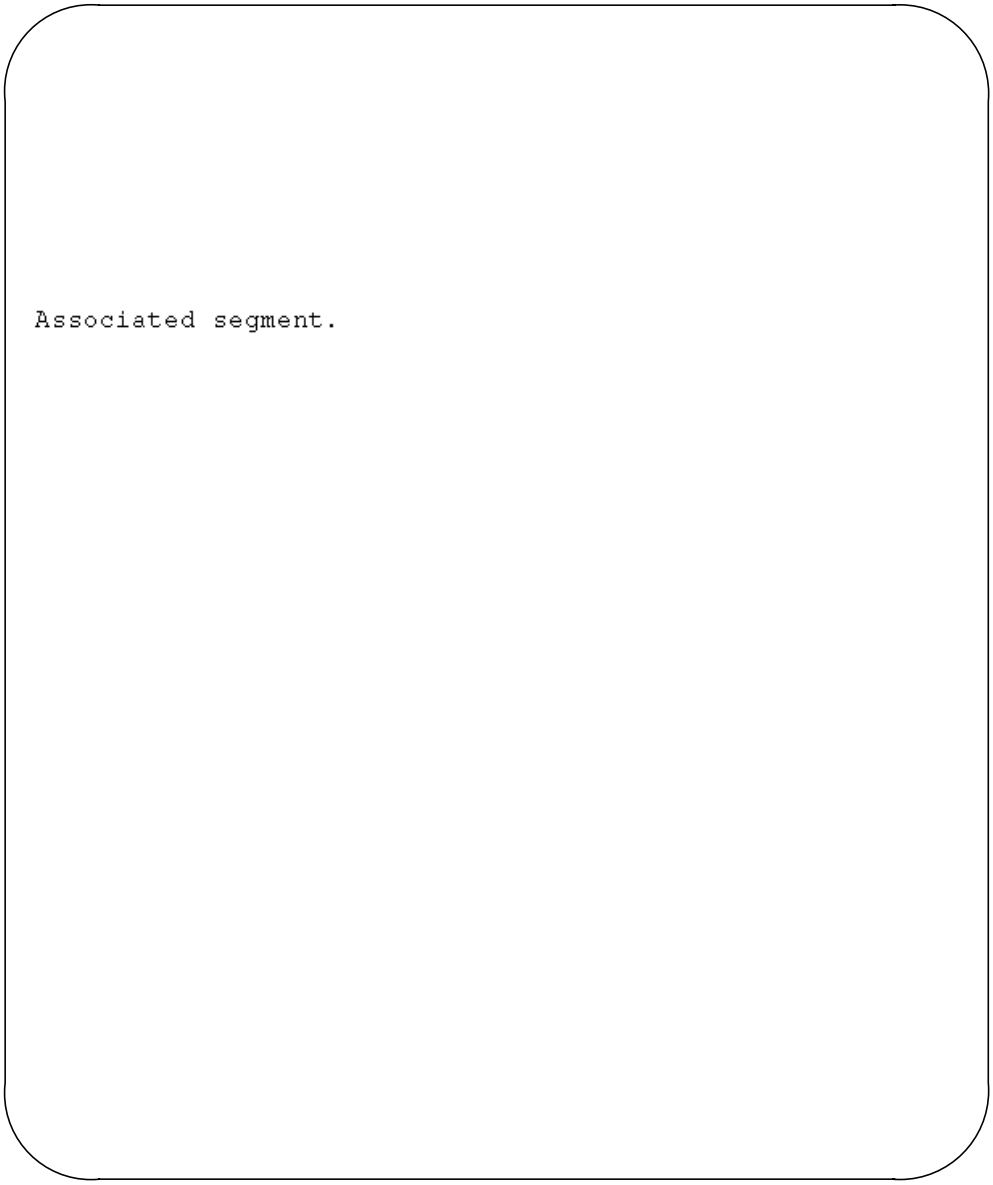
**Figure 8–1 Output with Two Segments**




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**

```
/*
 *  This program illustrates the use of the function INSERT SEGMENT.
 *  It draws a house in the lower left corner of the screen and then
 *  inserts that house into other segments.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

#  include <stdio.h>
#  include <gks.h>          /* C binding definition file */

main ()
{
    Gconn         conn_id       = GWC_DEF;
    Gcsw          coord_switch  = GNDC;
    Gevent        event;
    Gpoint        fixed_point;
    Gint          house_1       = 1;
    Gint          house_2       = 2;
    Gint          lower_left_corner = 1;
    Gint          num_pts       = 9;
    Gpoint        points[9];
    Gfloat        rotation      = 0;
    Gscale        scale;
    Gpoint        shift;
    Gfloat        time_out      = 5.00;
    Glimit        viewport;
    Gint          wiss          = 2;
    Gint          ws_id         = 1;
    Gwstype       ws_type       = GWS_DEF;
    Gwstype       ws_type_wiss  = GWS_WISS;
    Gfloat        xform_matrix[2][3];

/*  Open and activate GKS and the workstation environments. */

    gopengks (0, 0);
    gopenws (ws_id, &conn_id, &ws_type);
    gopenws (wiss, &conn_id, &ws_type_wiss);
    gactivatews (ws_id);
    gactivatews (wiss);

/*
 *  Set the viewport limits for the normalization transformation.
 *  The normalization window is established to be the lower left
 *  corner of NDC space.
 */

    viewport.xmin = viewport.ymin = 0.0;
    viewport.xmax = viewport.ymax = 0.5;

    gsetviewport (lower_left_corner, &viewport);
```

(continued on next page)

**Example 8–2 (Cont.) Inserting a Segment's Primitives into Another Segment**

```
/*  Create a segment in the lower left corner of the surface. */

    points[0].x  =  0.4;    points[0].y  = 0.1;
    points[1].x  =  0.1;    points[1].y  = 0.1;
    points[2].x  =  0.1;    points[2].y  = 0.7;
    points[3].x  =  0.4;    points[3].y  = 0.7;
    points[4].x  =  0.25;   points[4].y  = 0.9;
    points[5].x  =  0.1;    points[5].y  = 0.7;
    points[6].x  =  0.4;    points[6].y  = 0.1;
    points[7].x  =  0.4;    points[7].y  = 0.7;
    points[8].x  =  0.1;    points[8].y  = 0.1;


    gcreateseg (house_1);
    gselntran (lower_left_corner);
    gpolyline (num_pts, points);
    gcloseseg ();

/*  Deactivate WISS so no other segments are stored there. */

    gdeactivatews (wiss);

/*  Turn off the clipping so the transformed houses are visible. */

    gsetclip (GNOCLIP);

/*  Change the matrix value. */

    fixed_point.x = 0.0;    fixed_point.y = 0.0;
    scale.x_scale = 1.0;    scale.y_scale = 1.0;
    shift.x       = 0.5;    shift.y       = 0.0;

    gevaltran (&fixed_point, &shift, rotation, &scale, coord_switch,
       xform_matrix);

/*
 *  Create a segment in the lower right corner by inserting the primitives
 *  for house_1 into house_2.
 */

    gcreateseg (house_2);
    ginsertseg (house_1, xform_matrix);
    gcloseseg ();

/*
 *  Using EVALUATE TRANSFORMATION MATRIX, you can create the transformation
 *  matrix that you need to pass to INSERT SEGMENT as an argument.  This
 *  matrix specifies a position translation of 0.5 NDC points to the right.
 *  When this matrix is passed to INSERT SEGMENT while a segment is open,
 *  the house's primitives are transformed and made a part of the open
 *  segment.
 */

/*  Change the matrix value. */

    shift.x       = 0.0;    shift.y     = 0.5;

    gevaltran (&fixed_point, &shift, rotation, &scale, coord_switch,
       xform_matrix);
```

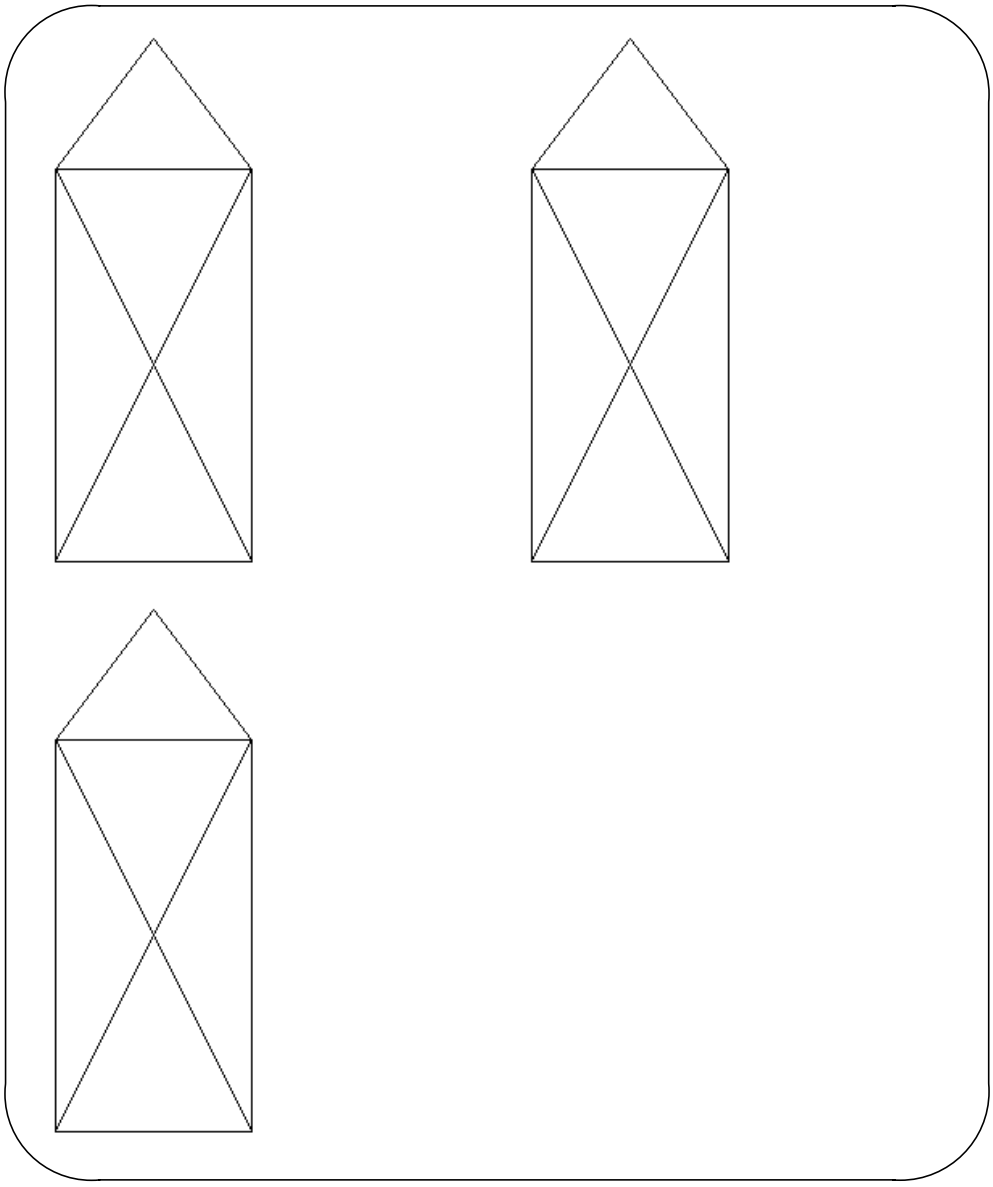
(continued on next page)

**Example 8–2 (Cont.) Inserting a Segment's Primitives into Another Segment**

```
/*
 *  Insert the primitives in the upper left corner using INSERT SEGMENT.
 *  Inserting segments when the GKS operating state is GWSAC causes
 *  the output primitives to be written to the workstation surface, but
 *  the primitives are not stored in a segment.  These segment primitives
 *  are translated 0.5 NDC points in an upwards direction.
 */

    ginsertseg (house_1, xform_matrix);

/*  Change the matrix value.  */

    shift.x       = 0.5;    shift.y     = 0.5;

    gevaltran (&fixed_point, &shift, rotation, &scale, coord_switch,
       xform_matrix);

/*  Insert the primitives in the upper right corner using INSERT SEGMENT. */

    ginsertseg (house_1, xform_matrix);

/*  Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPOSTPONE);
    gawaitevent (time_out, &event);

/*
 *  The call to REDRAW ALL SEGMENTS ON WORKSTATION redraws all segments
 *  and deletes all primitives outside of segments.  Wait 5 seconds.
 */

    gredrawsegws (ws_id);
    gawaitevent (time_out, &event);

/*  Deactivate and close the workstation environments and GKS. */

    gdeactivatews (ws_id);
    gclosews (wiss);
    gclosews (ws_id);
    gclosegks ();

}
```

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.

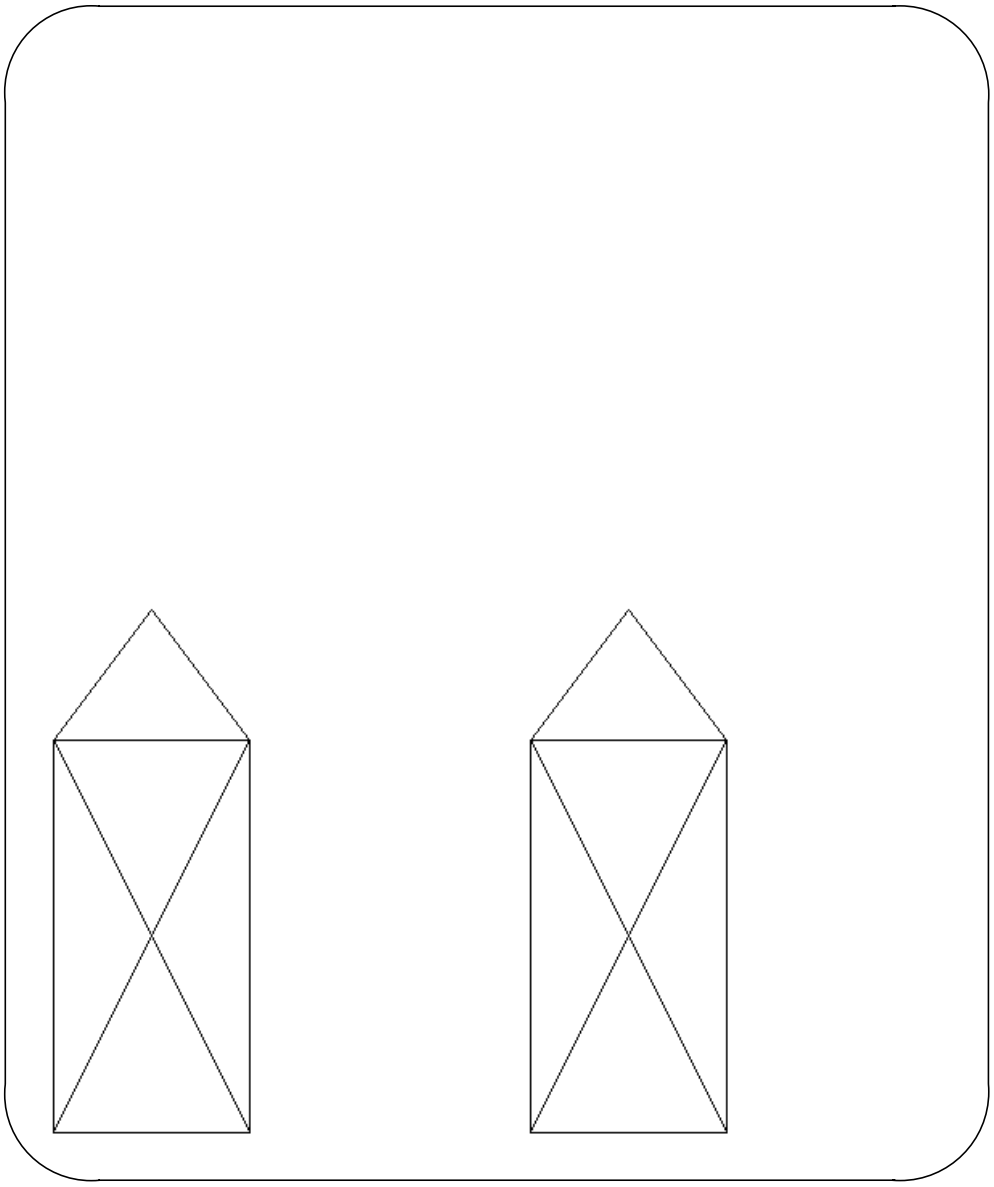
**Figure 8–3 Output of Original and Inserted Segments**



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**

```
/*
 *  This program illustrates the SET HIGHLIGHTING function.
 *  It draws a house in the lower left corner of the screen and a
 *  highlighted house in the upper right corner.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

#  include <stdio.h>
#  include <gks.h>       /* C binding definitions file */

main ()
{
    Gconn         conn_id            = GWC_DEF;
    Gevent        event;
    Gint          house_1            = 1;
    Gint          house_2            = 2;
    Gint          lower_left_corner  = 1;
    Gint          num_pts            = 9;
    Gpoint        points[9];
    Gfloat        time_out           = 5.00;
    Gint          upper_right_corner = 2;
    Glimit        viewport_1;
    Glimit        viewport_2;
    Gint          ws_id              = 1;
    Gwstype       ws_type            = GWS_DEF;

/*  Open and activate GKS and the workstation environment. */

    gopengks (0, 0);
    gopenws (ws_id, &conn_id, &ws_type);
    gactivatews (ws_id);

/*  Set the viewport limits for the normalization transformations. */

    viewport_1.xmin = viewport_1.ymin = 0.0;
    viewport_1.xmax = viewport_1.ymax = 0.5;

    gsetviewport (lower_left_corner, &viewport_1);

    viewport_2.xmin = viewport_2.ymin = 0.5;
    viewport_2.xmax = viewport_2.ymax = 1.0;

    gsetviewport (upper_right_corner, &viewport_2);

/*  Create a segment in the lower left corner of the surface. */

    points[0].x  =  0.4;    points[0].y  = 0.1;
    points[1].x  =  0.1;    points[1].y  = 0.1;
    points[2].x  =  0.1;    points[2].y  = 0.7;
    points[3].x  =  0.4;    points[3].y  = 0.7;
    points[4].x  =  0.25;   points[4].y  = 0.9;
    points[5].x  =  0.1;    points[5].y  = 0.7;
    points[6].x  =  0.4;    points[6].y  = 0.1;
    points[7].x  =  0.4;    points[7].y  = 0.7;
    points[8].x  =  0.1;    points[8].y  = 0.1;


    gcreateseg (house_1);
    gselntran (lower_left_corner);
    gpolyline (num_pts, points);
    gcloseseg ();
```

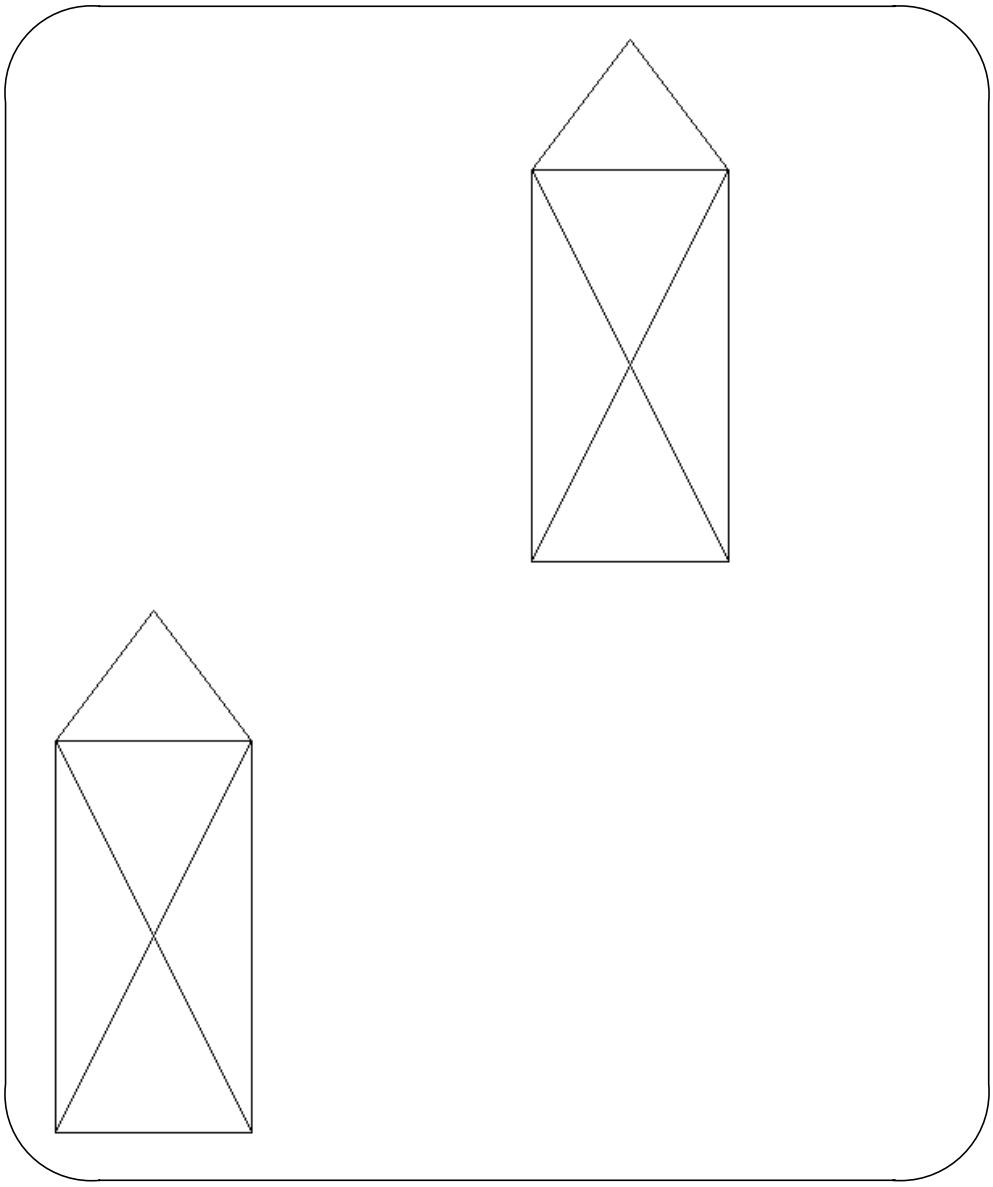
(continued on next page)

**Example 8–3 (Cont.) Highlighting a Segment**

```
/*  Create a second segment in the upper right corner of the surface. */

    gcreateseg (house_2);
    gselntran (upper_right_corner);
    gpolyline (num_pts, points);
    gcloseseg ();

/*  Release the deferred output.  Wait 5 seconds. */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*  Highlight house_2. */

    gsethighlight (house_2, GHIGHLIGHTED);

/*  Update the surface to initiate the change.  Wait 5 seconds. */

    gupdatews (ws_id, GPERFORM);
    gawaitevent (time_out, &event);

/*  Deactivate and close the workstation environment and GKS. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();

}
```

Figure 8–5 illustrates the houses before highlighting occurs.

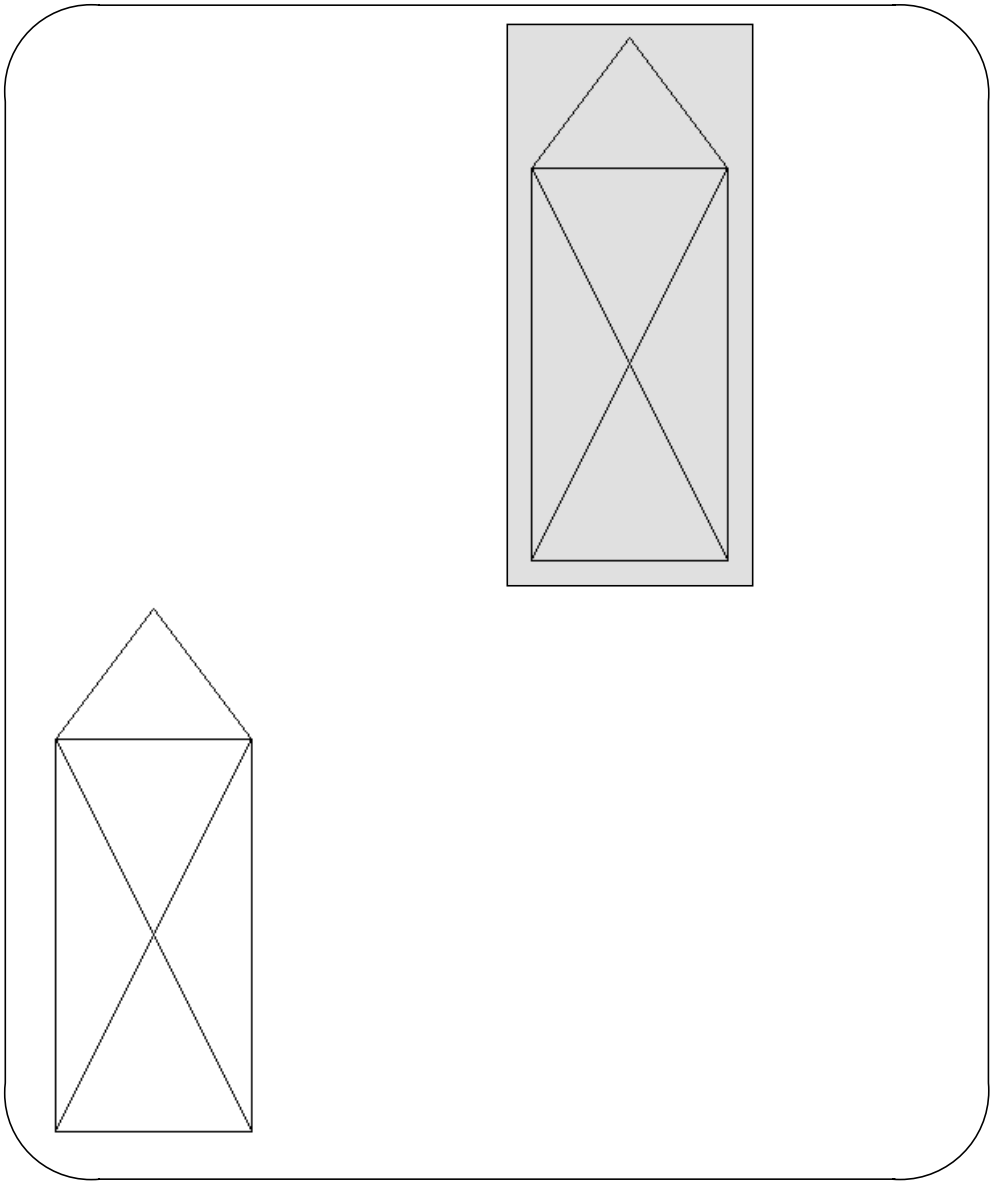
**Figure 8–5 Output Prior to Highlighting**



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 GNOECHO)

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.

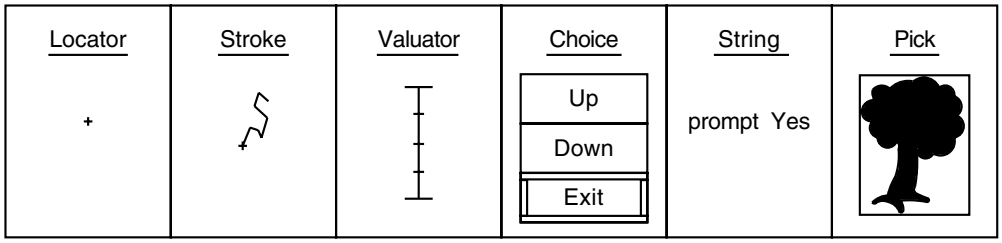
A choice-class logical input device creates a picture on the workstation surface that lists a series of choices. The choices are represented internally by integer values. You can label the choices with text in your application. For several workstations, the choices can look like a menu, with the currently 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.

The following sections list all the prompt and echo data record information for the C binding. The tables within these sections include the prompt and echo type, the input data record information, and whether the workstation uses (U) or ignores (I) the input data record. NA in the column means the information is not applicable.

Use the tables to select the prompt and echo type for an input class and to determine which data record information to supply to the workstation. Then use the C binding data structures described with the appropriate input class INITIALIZE function to pass the data record information to the workstation. The synchronous and asynchronous examples in the *DEC GKS User's Guide* demonstrate the initialization of several different logical input devices.

#### 9.3.2.1 Choice Class

This section lists the input data record information required for choice-class prompt and echo types.

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| –1, 1 | Number of choice alternatives | U |
| | Address of array of choice string lengths | U |
| | Address of array of choice string addresses | U |
| | Title string | U |
| 2 | Number of choice alternatives | U |
| | Address of array of prompts turned off (GPROFF) or on (GPRON) | U |
| 3 | Number of choice alternatives | U |
| | Address of array of choice string addresses | U |
| | Title string | U |

#### 9.3.2.2 Locator Class

This section lists the input data record information required for locator-class prompt and echo types.

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| –1 | X dimension of the box in WC coordinates. | U |
| | Y dimension of the box in WC coordinates. | U |
| –11, 1, 2, 3 | These prompt and echo types require no input data record information. Use a null pointer for the input data record pointer. | NA |
| 6 | Title string | U |
| –12, –10, –9, –5, –4, 4 | Attribute control flag tells the workstation to use either the current output attribute (GCURRENT) or the newly specified attributes provided in the data record (GSPECIFIED). | U |

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| | Line type ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Line width scale factor ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Polyline color index ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Polyline index. | I |
| | Line type index. | U[1] |
| | Line width scale factor. | U[1] |
| | Polyline color index. | I |
| –2, 5 | Polyline-fill-area control flag tells the workstation to use either a polyline (GPF_POLYLINE) or a fill area (GPF_FILLAREA) to draw the rectangle. Use GPF_POLYLINE because the fill area rectangle is not currently available. | I[2] |
| | Attribute control flag tells the workstation to use either the current output attribute (GCURRENT) or the newly specified attributes provided in the data record (GSPECIFIED). | U |
| | Line type ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Line width scale factor ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Polyline color index ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Polyline index. | I |
| | Line type index. | U[1] |
| | Line width scale factor. | U[1] |
| | Polyline color index. | I |
| | Fill area interior style ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Fill area style index ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Fill area color index ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Fill area index ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Fill area interior style (GHOLLOW, GSOLID, GPATTERN, or GHATCH). | I |
| | Fill area style index. | I |
| | Fill area color index. | I |

[1]If the attribute control flag is GSPECIFIED, the workstation uses the information. If the attribute control flag is GCURRENT, the workstation ignores the information.

[2]The workstation ignores this information because DEC GKS supports only the polyline rectangle. The workstation expects the flag to be GPF_POLYLINE.

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| –8, –7, –6, –3 | Attribute control flag tells the workstation to use either the current output attribute (GCURRENT) or the newly specified attributes provided in the data record (GSPECIFIED). | 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 |
| | Line type ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Line width scale factor ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Polyline color ASF (GBUNDLED or GINDIVIDUAL). | I |
| | Line type index. | U[1] |
| | Line width scale factor. | U[1] |
| | Polyline color index. | I |
| 13 | Segment identifier of the segment used for the cursor segment. | U |

[1]If the attribute control flag is GSPECIFIED, the workstation uses the information. If the attribute control flag is GCURRENT, the workstation ignores the information.

#### 9.3.2.3 Pick Class

This section lists the input data record information required for pick-class prompt and echo types.

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| 1, 2, 3 | Size of the pick aperture (prompt) in device coordinates | U |

#### 9.3.2.4 String Class

This section lists the input data record information required for string-class prompt and echo types.

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| 1 | Number of characters in the input buffer | U |
| | Initial cursor position within the string, 1 <= position <= string_length | I |
| | Title string | U |

#### 9.3.2.5 Stroke Class

This section lists the input data record information required for stroke-class prompt and echo types.

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| 1 | Number of stroke points in the input buffer | U |
| | Editing position expressed as a stroke point | I |
| | X component of the WC change vector | U |
| | Y component of the WC change vector | U |
| | Time interval, in seconds | I |
| 3 | Number of stroke points in the input buffer | U |
| | Editing position expressed as a stroke point | I |
| | X component of the WC change vector | U |
| | Y component of the WC change vector | U |
| | Time interval, in seconds | I |
| | Attribute control flag tells the workstation to use either the current output attribute (GCURRENT) or the newly specified attributes provided in the data record (GSPECIFIED) | U |
| | Polymarker type ASF (GBUNDLED or GINDIVIDUAL) | I |
| | Polymarker size factor ASF (GBUNDLED or GINDIVIDUAL) | I |
| | Polymarker color ASF (GBUNDLED or GINDIVIDUAL) | I |
| | Polymarker bundle index | I |
| | Polymarker type index | U[1] |
| | Polymarker scale factor | U[1] |
| | Polymarker color index | I |
| 4 | Number of stroke points in the input buffer | U |
| | Editing position expressed as a stroke point | I |
| | X component of the WC change vector | U |
| | Y component of the WC change vector | U |
| | Time interval, in seconds | I |
| | Attribute control flag tells the workstation to use either the current output attribute (GCURRENT) or the newly specified attributes provided in the data record (GSPECIFIED) | U |
| | Line type ASF (GBUNDLED or GINDIVIDUAL) | I |
| | Line width scale factor ASF (GBUNDLED or GINDIVIDUAL) | I |

[1]If the attribute control flag is GSPECIFIED, the workstation uses the information. If the attribute control flag is GCURRENT, the workstation ignores the information.

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| | Polyline color index ASF (GBUNDLED or GINDIVIDUAL) | I |
| | Polyline index | I |
| | Line type index | U[1] |
| | Line width scale factor | U[1] |
| | Polyline color index | I |

[1]If the attribute control flag is GSPECIFIED, the workstation uses the information. If the attribute control flag is GCURRENT, the workstation ignores the information.

#### 9.3.2.6 Valuator Class

This section lists the input data record information required for valuator-class prompt and echo types.

| Prompt and Echo Type | Input Data Record Information | Used or Ignored |
|---|---|---|
| –3, –2, –1, 1, 2, 3 | Low value of the numeric range | U |
| | High value of the numeric range | U |
| | Title string | U |

## 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 GC_OK to the request function *response* 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 *response* 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, GC_NOCHOICE or GP_NOPICK, to the *response* argument. (DEC GKS also returns GP_NOPICK 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: GP_NOCHOICE or GP_NOPICK. You can obtain the value GP_NOCHOICE if the user did not alter the device's measure since it was activated. You can obtain the value GP_NOPICK 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 *response* 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 *timeout* 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 GNCLASS to its *event* 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 *event* 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 *error* 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 *error* does not equal 0, the information needed to write to the output arguments is not available. In this case, *error* 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 GNCLASS.

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 on the NDC plane. 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 variables to 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)

Be careful when passing the argument containing the data record buffer size to the inquiry functions. The buffer size is a modifiable variable (read/write). When passed to the inquiry function, the argument must contain the size of the buffer. If it does not, the inquiry function will not return the contents of the data record properly.

After the function call, DEC GKS writes the amount of the buffer actually used. You can compare this value to the data record buffer size to see if DEC GKS had to truncate the data record when writing it to the buffer. If DEC GKS truncated the data record, you must decide whether to continue execution or change the buffer size so the entire data record fits.

### 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

gawaitevent (

```
  Gfloat   timeout,    /* (I) Maximum wait period (seconds) */
  Gevent   *event      /* (O) Event workstation, class, and device
                              number */
```

)

## Data Structures

```
typedef struct {     /* EVENT */
  Gint      ws;      /* workstation */
  Gint      dev;     /* device number */
  Giclass   class;   /* input class (constant) */
} Gevent;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Giclass | GNCLASS | Input queue is empty |
| | GLOCATOR | Event from a locator device |
| | GSTROKE | Event from a stroke device |
| | GVALUATOR | Event from a valuator device |
| | GCHOICE | Event from a choice device |
| | GPICK | Event from a pick device |
| | GSTRING | Event from a string device |
| | GVIEWPORT | 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 GNCLASS 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 GNCLASS 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

gflushevents (

```
  Gint       ws,          /* (I) Workstation identifier */
  Giclass    class,       /* (I) Input device class (constant) */
  Gint       dev          /* (I) Logical input device number */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Giclass | GNCLASS | Input queue is empty |
| | GLOCATOR | Event from a locator device |
| | GSTROKE | Event from a stroke device |
| | GVALUATOR | Event from a valuator device |
| | GCHOICE | Event from a choice device |
| | GPICK | Event from a pick device |
| | GSTRING | Event from a string device |
| | GVIEWPORT | 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

ggetchoice (

```
  Gchoice     *response      /* (O) Status and choice number */
```

)

## Data Structures

```
typedef struct {        /* CHOICE DATA */
  Gcstat    status;     /* choice status (constant) */
  Gint      choice;     /* choice number */
} Gchoice;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcstat | GC_OK | Input obtained |
| | GC_NOCHOICE | Triggered without choosing |

## 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 GC_OK or GC_NOCHOICE.

## See Also

AWAIT EVENT
FLUSH DEVICE EVENTS
SET CHOICE MODE

# GET LOCATOR

## Operating States

WSOP, WSAC, SGOP

## Syntax

ggetloc (

```
  Gloc    *response    /* (O) Normalization transformation and location */
```

)

## Data Structures

```
typedef struct {          /* LOCATOR DATA */
  Gint      transform;    /* normalization transformation number */
  Gpoint    position;     /* locator position */
} Gloc;

    typedef struct {      /* COORDINATE POINT */
      Gfloat    x;        /* X coordinate */
      Gfloat    y;        /* Y coordinate */
    } Gpoint;
```

## 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

ggetloc3 (

```
  Gloc3    *response     /* (O) Status, transformation, index, and
                              location */
```

)

## Data Structures

```
typedef struct {          /* LOCATOR 3 DATA */
  Gint      transform;    /* normalization transformation number */
  Gint      view;         /* view index */
  Gpoint3   position;     /* locator position */
} Gloc3;

    typedef struct {      /* COORDINATE POINT */
      Gfloat    x;        /* X coordinate */
      Gfloat    y;        /* Y coordinate */
      Gfloat    z;        /* Z coordinate */
    } Gpoint3;
```

## 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

ggetpick (

```
  Gpick   *response    /* (O) Status, segment identifier, and
                              pick identifier */
```

)

## Data Structures

```
typedef struct {       /* PICK DATA */
  Gpstat    status;    /* pick status (constant) */
  Gint      seg;       /* pick segment */
  Gint      pickid;    /* pick identifier */
} Gpick;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gpstat | GP_OK | Input obtained |
| | GP_NOPICK | 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

ggetstring (

```
  Gchar   *response    /* (O) Returned character string.  Initialize
                           this argument with a string, before the
                           call.  The string buffer you give to this
                           call should be at least one more than the
                           maximum possible string size.  It is best
                           to specify the string buffer size as 256
                           characters. */
```

)

### Description

The GET STRING function obtains information from the *current event report* entry in the GKS state list and writes the string to the string output argument.

When activating string input, 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 1 byte larger than the maximum possible string size.
- 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

ggetstroke (

```
  Gstroke     *response     /* (0) Status, transformation, and stroke */
```

)

## Data Structures

```
typedef struct {         /* STROKE DATA */
  Gint      transform;   /* normalization transformation number */
  Gint      n_points;    /* number of points in stroke */
  Gpoint    *points;     /* points in stroke */
} Gstroke;

    typedef struct {     /* COORDINATE POINT */
      Gfloat    x;       /* X coordinate */
      Gfloat    y;       /* Y coordinate */
    } Gpoint;
```

**Note**

The field *n_points* should be initialized with the number representing the maximum number of points that will fit in the points buffer. It will be updated to indicate the actual number of points returned in the buffer.

## 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_points* field.
- 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

ggetstroke3 (

```
  Gstroke3    *response    /* (O) Status, transformation, view,
                                  and stroke */
```

)

## Data Structures

```
typedef struct {          /* STROKE 3 DATA */
  Gint      transform;    /* normalization transformation number */
  Gint      view;         /* view index used in transformation */
  Gint      n_points;     /* number of points in stroke */
  Gpoint3   *points;      /* points in stroke */
} Gstroke3;

    typedef struct {      /* COORDINATE POINT */
      Gfloat    x;        /* X coordinate */
      Gfloat    y;        /* Y coordinate */
      Gfloat    z;        /* Z coordinate */
    } Gpoint3;
```

**Note**

The field *n_points* should be initialized with the number representing the maximum number of points that will fit in the points buffer. It will be updated to indicate the actual number of points returned in the buffer.

## 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_points* field.
- 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

ggetval (

```
  Gfloat     *response        /* (O) Returned value */
```

)

### 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

ginitchoice (

```
  Gint          ws,           /* (I) Workstation identifier */
  Gint          dev,          /* (I) Choice device number */
  Gchoice       *init,        /* (I) Initial status and choice */
  Gint          pet,          /* (I) Prompt and echo type */
  Glimit        *area,        /* (I) Echo area */
  Gchoicerec    *data         /* (I) Choice data record */
```

)

## Data Structures

```
typedef struct {         /* CHOICE DATA */
  Gcstat      status;    /* choice status (constant) */
  Gint        choice;    /* choice number */
} Gchoice;

typedef struct {         /* COORDINATE LIMITS */
  Gfloat      xmin;      /* X minimum limit */
  Gfloat      xmax;      /* X maximum limit */
  Gfloat      ymin;      /* Y minimum limit */
  Gfloat      ymax;      /* Y maximum limit */
} Glimit;

typedef union {         /* CHOICE DATA RECORD */
  Gchoicepet_0001       choicepet_1_datarec;
  Gchoicepet0001        choicepet1_datarec;
  Gchoicepet0002        choicepet2_datarec;
  Gchoicepet0003        choicepet3_datarec;
  Gchoicepet0004        choicepet4_datarec;
  Gchoicepet0005        choicepet5_datarec;
} Gchoicerec;

    typedef Gchoicepetneg0001 Gchoicepet_0001;

        typedef Gchoicepet0001  Gchoicepetneg0001;

        typedef struct {
          Gint      number;         /* number of choice strings */
          Gint      *lengths;       /* lengths of choice strings */
          Gchar     **strings;      /* array of strings */
          Gchar     *title_string;  /* the title string */
        } Gchoicepet0001;

    typedef struct {
      Gint    number;        /* number of alternatives */
      Gprflag *enable;       /* array of prompts */
      Gchar   *title_string; /* title string */
    } Gchoicepet0002;

    typedef struct {
      Gint    number;        /* number of choice strings */
      Gchar   **strings;     /* array of strings */
      Gchar   *title_string; /* the title string */
    } Gchoicepet0003;
```

```
typedef Gchoicepet0003 Gchoicepet0004;

typedef struct {
  Gint    seg;           /* segment name */
  Gint    number;        /* number of alternatives */
  Gint    *pickids;      /* array of pick identifiers */
  Gchar   *title_string; /* the title string */
} Gchoicepet0005;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcstat | GC_OK | Input obtained |
| | GC_NOCHOICE | Triggered without choosing |
| Gprflag | GPROFF | Choice input prompt flag off. |
| | GPRON | Choice input prompt flag on. |

## 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

SET CHOICE MODE
Example 9–3 for a program example using an INITIALIZE . . . function

# INITIALIZE CHOICE 3

## Operating States

WSOP, WSAC, SGOP

## Syntax

ginitchoice3 (

```
  Gint          ws,         /* (I) Workstation identifier */
  Gint          dev,        /* (I) Choice device number */
  Gchoice       *init,      /* (I) Initial status and choice */
  Gint          pet,        /* (I) Prompt and echo type */
  Glimit3       *volume,    /* (I) Echo volume */
  Gchoicerec    *data       /* (I) Choice data record */
```

)

## Data Structures

```
typedef struct {          /* CHOICE DATA */
  Gcstat       status;    /* choice status (constant) */
  Gint         choice;    /* choice number */
} Gchoice;

typedef struct {          /* COORDINATE LIMITS */
  Gfloat       xmin;      /* X minimum limit */
  Gfloat       xmax;      /* X maximum limit */
  Gfloat       ymin;      /* Y minimum limit */
  Gfloat       ymax;      /* Y maximum limit */
  Gfloat       zmin;      /* Z minimum limit */
  Gfloat       zmax;      /* Z maximum limit */
} Glimit3;

typedef union {         /* CHOICE DATA RECORD */
  Gchoicepet_0001       choicepet_1_datarec;
  Gchoicepet0001        choicepet1_datarec;
  Gchoicepet0002        choicepet2_datarec;
  Gchoicepet0003        choicepet3_datarec;
  Gchoicepet0004        choicepet4_datarec;
  Gchoicepet0005        choicepet5_datarec;
} Gchoicerec;

    typedef Gchoicepetneg0001 Gchoicepet_0001;

        typedef Gchoicepet0001  Gchoicepetneg0001;

        typedef struct {
          Gint      number;         /* number of choice strings */
          Gint      *lengths;       /* lengths of choice strings */
          Gchar     **strings;      /* array of strings */
          Gchar     *title_string;  /* the title string */
        } Gchoicepet0001;

    typedef struct {
      Gint    number;        /* number of alternatives */
      Gprflag *enable;       /* array of prompts */
      Gchar   *title_string; /* title string */
    } Gchoicepet0002;
```

```
typedef struct {
  Gint    number;        /* number of choice strings */
  Gchar   **strings;     /* array of strings */
  Gchar   *title_string; /* the title string */
} Gchoicepet0003;

typedef Gchoicepet0003 Gchoicepet0004;

typedef struct {
  Gint    seg;           /* segment name */
  Gint    number;        /* number of alternatives */
  Gint    *pickids;      /* array of pick identifiers */
  Gchar   *title_string; /* the title string */
} Gchoicepet0005;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcstat | GC_OK | Input obtained |
| | GC_NOCHOICE | Triggered without choosing |
| Gprflag | GPROFF | Choice input prompt flag off. |
| | GPRON | Choice input prompt flag on. |

## 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

SET CHOICE MODE
Example 9–3 for a program example using an INITIALIZE . . . function

# INITIALIZE LOCATOR

## Operating States

WSOP, WSAC, SGOP

## Syntax

ginitloc (

```
  Gint      ws,         /* (I) Workstation identifier */
  Gint      dev,        /* (I) Locator device number */
  Gloc      *init,      /* (I) Initial transformation, view, and location */
  Gint      pet,        /* (I) Prompt and echo type */
  Glimit    *area,      /* (I) Echo area */
  Glocrec   *data       /* (I) Locator data record */
```

)

## Data Structures

```
typedef struct {         /* LOCATOR DATA */
  Gint      transform;   /* normalization transformation number */
  Gpoint    position;    /* locator position */
} Gloc;

    typedef struct {     /* COORDINATE POINT */
      Gfloat    x;       /* X coordinate */
      Gfloat    y;       /* Y coordinate */
    } Gpoint;

typedef struct {         /* COORDINATE LIMITS */
  Gfloat    xmin;        /* X minimum limit */
  Gfloat    xmax;        /* X maximum limit */
  Gfloat    ymin;        /* Y minimum limit */
  Gfloat    ymax;        /* Y maximum limit */
} Glimit;

typedef union {         /* LOCATOR DATA RECORD */
  Glocpet_0001          locpet_1_datarec;
  Glocpet_0002          locpet_2_datarec;
  Glocpet_0003          locpet_3_datarec;
  Glocpet_0004          locpet_4_datarec;
  Glocpet_0005          locpet_5_datarec;
  Glocpet_0006          locpet_6_datarec;
  Glocpet_0007          locpet_7_datarec;
  Glocpet_0008          locpet_8_datarec;
  Glocpet_0009          locpet_9_datarec;
  Glocpet_0010          locpet_10_datarec;
  Glocpet_0011          locpet_11_datarec;
  Glocpet_0012          locpet_12_datarec;
  Glocpet0001           locpet1_datarec;
  Glocpet0002           locpet2_datarec;
  Glocpet0003           locpet3_datarec;
  Glocpet0004           locpet4_datarec;
  Glocpet0005           locpet5_datarec;
  Glocpet0006           locpet6_datarec;
} Glocrec;

    typedef Glocpetneg0001  Glocpet_0001;
```

```
    typedef struct {
      Gfloat   box_x;          /* size of the box in x */
      Gfloat   box_y;          /* size of the box in y */
      Gchar    *data;          /* device/implementation dependent data */
    } Glocpetneg0001;

typedef Glocpetneg0002  Glocpet_0002;

    typedef Glocpet0005 Glocpetneg0002;

        typedef struct {
          Gpfcf      pfcf;         /* polyline/fill area control flag
                                      (constant) */
          Gacf       acf;          /* attribute control flag */
          union {
                Glnattr    ln;     /* polyline attributes */
                Gflattr    fl;     /* fill area attributes */
          } attr;
          Gchar      *data;        /* device/implementation dependent
                                      data */
        } Glocpet0005;

typedef Glocpetneg0003  Glocpet_0003;

    typedef struct {
      Gacf   acf;              /* attribute control flag (constant) */
      union {
          struct {
             Gpoint   point1;  /* point 1 for echo */
             Gpoint   point2;  /* point 2 for echo */
          } echo;
          struct
             Glnattr  ln;      /* polyline attributes */
             Gpoint   point1;  /* point 1 for echo */
             Gpoint   point2;  /* point 2 for echo */
          } lnecho;
      } attr;
      Gchar  *data;            /* device/implementation dependent data */
    } Glocpetneg0003;

        typedef struct {      /* POLYLINE ATTRIBUTES */
          Gasf      type;     /* line type ASF (constant) */
          Gasf      width;    /* line width ASF */
          Gasf      colour;   /* line color ASF */
          Gint      line;     /* line index */
          Glnbundl  bundl;    /* line bundle */
        } Glnattr;

            typedef struct {       /* POLYLINE BUNDLE */
              Gint      type;    /* line type (constant) */
              Gfloat    width;   /* linewidth scale factor */
              Gint      colour;  /* polyline colour index */
            } Glnbundl;

typedef Glocpetneg0004  Glocpet_0004;

    typedef struct {
      Gacf       acf;     /* attribute control flag */
      Glnattr    ln;      /* polyline attributes */
      Gchar      *data;   /* device/implementation dependent data */
    } Glocpetneg0004;

typedef Glocpetneg0005  Glocpet_0005;

    typedef Glocpetneg0004 Glocpetneg0005;

typedef Glocpetneg0006  Glocpet_0006;

    typedef Glocpetneg0003  Glocpetneg0006;

typedef Glocpetneg0007  Glocpet_0007;
```

```
    typedef Glocpetneg0003  Glocpetneg0007;

typedef Glocpetneg0008  Glocpet_0008;

    typedef Glocpetneg0003 Glocpetneg0008;

typedef Glocpetneg0009  Glocpet_0009;

    typedef Glocpetneg0004  Glocpetneg0009;

typedef Glocpetneg0010  Glocpet_0010;

     typedef Glocpetneg0004 Glocpetneg0010;

typedef Glocpetneg0011  Glocpet_0011;

    typedef Glocpet0001 Glocpetneg0011;

        typedef struct {     /* LOCATOR prompt and echo types */
          Gchar   *data;     /* device/implementation dependent data */
        } Glocpet0001;

typedef Glocpetneg0012  Glocpet_0012;

    typedef Glocpetneg0004  Glocpetneg0012;

typedef Glocpet0001 Glocpet0002;

typedef Glocpet0001 Glocpet0003;

typedef struct {
    Gacf       acf;          /* attribute control flag */
    Glnattr    ln;           /* polyline attributes */
    Gchar      *data;        /* device/implementation dependent data */
} Glocpet0004;

typedef struct {
  Gchar    *title_string;   /* the title string */
} Glocpet0006;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gacf | GCURRENT | Input data record current values |
| | GSPECIFIED | Input data record specified values |
| Gasf | GBUNDLED | Bundled attributes |
| | GINDIVIDUAL | Individual attributes |
| Line types | GLN_SOLID | Solid line |
| | GLN_DASHED | Dashed line |
| | GLN_DOTTED | Dotted line |
| | GLN_DASHDOT | Dashed-dotted line |
| Gpfcf | GPF_POLYLINE | Data record polyline control flag |
| | GPF_FILLAREA | Data record fill area control flag |
| Gflinter | GHOLLOW | Hollow interior |
| | GSOLID | Solid interior |
| | GPATTERN | Patterned interior |
| | GHATCH | Hatched interior |

## INITIALIZE LOCATOR

### 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

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

ginitloc3 (

```
  Gint       ws,        /* (I) Workstation identifier */
  Gint       dev,       /* (I) Locator device number */
  Gloc3      *init,     /* (I) Initial transformation, view, and location */
  Gint       pet,       /* (I) Prompt and echo type */
  Glimit3    *volume,   /* (I) Echo volume */
  Glocrec    *data      /* (I) Locator data record */
```

)

## Data Structures

```
typedef struct {         /* LOCATOR 3 DATA */
  Gint      transform;   /* normalization transformation number */
  Gint      view;        /* view index */
  Gpoint3   position;    /* locator position */
} Gloc3;

    typedef struct {     /* COORDINATE POINT */
      Gfloat    x;       /* X coordinate */
      Gfloat    y;       /* Y coordinate */
      Gfloat    z;       /* Z coordinate */
    } Gpoint3;

typedef struct {         /* COORDINATE LIMITS */
  Gfloat    xmin;        /* X minimum limit */
  Gfloat    xmax;        /* X maximum limit */
  Gfloat    ymin;        /* Y minimum limit */
  Gfloat    ymax;        /* Y maximum limit */
  Gfloat    zmin;        /* Z minimum limit */
  Gfloat    zmax;        /* Z maximum limit */
} Glimit3;

typedef union {         /* LOCATOR DATA RECORD */
  Glocpet_0001         locpet_1_datarec;
  Glocpet_0002         locpet_2_datarec;
  Glocpet_0003         locpet_3_datarec;
  Glocpet_0004         locpet_4_datarec;
  Glocpet_0005         locpet_5_datarec;
  Glocpet_0006         locpet_6_datarec;
  Glocpet_0007         locpet_7_datarec;
  Glocpet_0008         locpet_8_datarec;
  Glocpet_0009         locpet_9_datarec;
  Glocpet_0010         locpet_10_datarec;
  Glocpet_0011         locpet_11_datarec;
  Glocpet_0012         locpet_12_datarec;
  Glocpet0001          locpet1_datarec;
  Glocpet0002          locpet2_datarec;
  Glocpet0003          locpet3_datarec;
  Glocpet0004          locpet4_datarec;
  Glocpet0005          locpet5_datarec;
  Glocpet0006          locpet6_datarec;
} Glocrec;
```

```
typedef Glocpetneg0001  Glocpet_0001;

    typedef struct {
      Gfloat   box_x;          /* size of the box in x */
      Gfloat   box_y;          /* size of the box in y */
      Gchar    *data;          /* device/implementation dependent data */
    } Glocpetneg0001;

typedef Glocpetneg0002  Glocpet_0002;

    typedef Glocpet0005 Glocpetneg0002;

        typedef struct {
          Gpfcf      pfcf;         /* polyline/fill area control flag
                                      (constant) */
          Gacf       acf;          /* attribute control flag */
          union {
                Glnattr    ln;     /* polyline attributes */
                Gflattr    fl;     /* fill area attributes */
          } attr;
          Gchar      *data;        /* device/implementation dependent
                                      data */
        } Glocpet0005;

typedef Glocpetneg0003  Glocpet_0003;

    typedef struct {
      Gacf   acf;              /* attribute control flag (constant) */
      union {
          struct {
             Gpoint   point1;  /* point 1 for echo */
             Gpoint   point2;  /* point 2 for echo */
          } echo;
          struct
             Glnattr  ln;      /* polyline attributes */
             Gpoint   point1;  /* point 1 for echo */
             Gpoint   point2;  /* point 2 for echo */
          } lnecho;
      } attr;
      Gchar  *data;            /* device/implementation dependent data */
    } Glocpetneg0003;

        typedef struct {      /* POLYLINE ATTRIBUTES */
          Gasf      type;     /* line type ASF (constant) */
          Gasf      width;    /* line width ASF */
          Gasf      colour;   /* line color ASF */
          Gint      line;     /* line index */
          Glnbundl  bundl;    /* line bundle */
        } Glnattr;

            typedef struct {       /* POLYLINE BUNDLE */
              Gint      type;    /* line type (constant) */
              Gfloat    width;   /* linewidth scale factor */
              Gint      colour;  /* polyline colour index */
            } Glnbundl;

typedef Glocpetneg0004  Glocpet_0004;

    typedef struct {
      Gacf       acf;     /* attribute control flag */
      Glnattr    ln;      /* polyline attributes */
      Gchar      *data;   /* device/implementation dependent data */
    } Glocpetneg0004;

typedef Glocpetneg0005  Glocpet_0005;

    typedef Glocpetneg0004 Glocpetneg0005;

typedef Glocpetneg0006  Glocpet_0006;

    typedef Glocpetneg0003  Glocpetneg0006;
```

```
typedef Glocpetneg0003 Glocpetneg0006;

typedef Glocpetneg0007  Glocpet_0007;

    typedef Glocpetneg0003  Glocpetneg0007;

typedef Glocpetneg0008  Glocpet_0008;

    typedef Glocpetneg0003 Glocpetneg0008;

typedef Glocpetneg0009  Glocpet_0009;

    typedef Glocpetneg0004  Glocpetneg0009;

typedef Glocpetneg0010  Glocpet_0010;

     typedef Glocpetneg0004 Glocpetneg0010;

typedef Glocpetneg0011  Glocpet_0011;

    typedef Glocpet0001 Glocpetneg0011;

        typedef struct {     /* LOCATOR prompt and echo types */
          Gchar   *data;     /* device/implementation dependent data */
        } Glocpet0001;

typedef Glocpetneg0012  Glocpet_0012;

    typedef Glocpetneg0004  Glocpetneg0012;

typedef Glocpet0001 Glocpet0002;

typedef Glocpet0001 Glocpet0003;

typedef struct {
  Gacf       acf;          /* attribute control flag */
  Glnattr    ln;           /* polyline attributes */
  Gchar      *data;        /* device/implementation dependent data */
} Glocpet0004;

typedef struct {
  Gchar    *title_string;   /* the title string */
} Glocpet0006;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gacf | GCURRENT | Input data record current values |
| | GSPECIFIED | Input data record specified values |
| Gasf | GBUNDLED | Bundled attributes |
| | GINDIVIDUAL | Individual attributes |
| Line types | GLN_SOLID | Solid line |
| | GLN_DASHED | Dashed line |
| | GLN_DOTTED | Dotted line |
| | GLN_DASHDOT | Dashed-dotted line |
| Gpfcf | GPF_POLYLINE | Data record polyline control flag |
| | GPF_FILLAREA | Data record fill area control flag |
| Gflinter | GHOLLOW | Hollow interior |
| | GSOLID | Solid interior |
| | GPATTERN | Patterned interior |
| | GHATCH | Hatched interior |

## INITIALIZE LOCATOR 3

### 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

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

ginitpick (

```
  Gint        ws,      /* (I) Workstation identifier */
  Gint        dev,     /* (I) Pick device number */
  Gpick       *init,   /* (I) Initial status, segment identifier, and
                          pick identifier */
  Gint        pet,     /* (I) Pick prompt and echo type */
  Glimit      *area,   /* (I) Echo area */
  Gpickrec    *data    /* (I) Pick data record */
```

)

## Data Structures

```
typedef struct {       /* PICK DATA */
  Gpstat    status;    /* pick status (constant) */
  Gint      seg;       /* pick segment */
  Gint      pickid;    /* pick identifier */
} Gpick;

typedef struct {    /* COORDINATE LIMITS */
  Gfloat    xmin;   /* X minimum limit */
  Gfloat    xmax;   /* X maximum limit */
  Gfloat    ymin;   /* Y minimum limit */
  Gfloat    ymax;   /* Y maximum limit */
} Glimit;

typedef union {     /* PICK DATA RECORD */
  Gpickpet0001       pickpet1_datarec;
  Gpickpet0002       pickpet2_datarec;
  Gpickpet0003       pickpet3_datarec;
} Gpickrec;

    typedef struct {
      Gfloat    aperture;   /* pick aperture in DC */
      Gchar     *data;      /* device/implementation dependent data */
    } Gpickpet0001;

    typedef Gpickpet0001 Gpickpet0002;

    typedef Gpickpet0001 Gpickpet0003;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gpstat | GP_OK | The initial segment and pick identifier are chosen. |
| | GP_NOPICK | No segment or pick identifier is returned. |

## INITIALIZE PICK

### 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

SET PICK MODE
Example 9–2 for a program example using the INITIALIZE PICK function

# INITIALIZE PICK 3

## Operating States

WSOP, WSAC, SGOP

## Syntax

ginitpick3 (

```
  Gint       ws,       /* (I) Workstation identifier */
  Gint       dev,      /* (I) Pick device number */
  Gpick      *init,    /* (I) Initial status, segment identifier, and
                           pick identifier */
  Gint       pet,      /* (I) Pick prompt and echo type */
  Glimit3    *volume,  /* (I) Echo volume */
  Gpickrec   *data     /* (I) Pick data record */
```

)

## Data Structures

```
typedef struct {       /* PICK DATA */
  Gpstat    status;    /* pick status (constant) */
  Gint      seg;       /* pick segment */
  Gint      pickid;    /* pick identifier */
} Gpick;

typedef struct {       /* COORDINATE LIMITS */
  Gfloat    xmin;      /* X minimum limit */
  Gfloat    xmax;      /* X maximum limit */
  Gfloat    ymin;      /* Y minimum limit */
  Gfloat    ymax;      /* Y maximum limit */
  Gfloat    zmin;      /* Z minimum limit */
  Gfloat    zmax;      /* Z maximum limit */
} Glimit3;

typedef union {        /* PICK DATA RECORD */
  Gpickpet0001         pickpet1_datarec;
  Gpickpet0002         pickpet2_datarec;
  Gpickpet0003         pickpet3_datarec;
} Gpickrec;
```

```
typedef struct {
  Gfloat    aperture;    /* pick aperture in DC */
  Gchar     *data;       /* device/implementation dependent data */
} Gpickpet0001;

typedef Gpickpet0001 Gpickpet0002;

typedef Gpickpet0001 Gpickpet0003;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gpstat | GP_OK | The initial segment and pick identifier are chosen. |
| | GP_NOPICK | No segment or pick identifier is returned. |

## INITIALIZE PICK 3

### 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

SET PICK MODE
Example 9–2 for a program example using the INITIALIZE PICK function

# INITIALIZE STRING

## Operating States

WSOP, WSAC, SGOP

## Syntax

ginitstring (

```
  Gint          ws,        /* (I) Workstation identifier. */
  Gint          dev,       /* (I) String device number. */
  Gchar         *init,     /* (I) Initial string. Once you request input,
                               the user can delete or edit the initial
                               string; otherwise, the newly input
                               string is appended to the initial
                               string. */
  Gint          pet,       /* (I) String prompt and echo type. */
  Glimit        *area,     /* (I) Echo area. */
  Gstringrec    *data      /* (I) String data record. */
```

)

## Data Structures

```
typedef struct {     /* COORDINATE LIMITS */
  Gfloat    xmin;    /* X minimum limit */
  Gfloat    xmax;    /* X maximum limit */
  Gfloat    ymin;    /* Y minimum limit */
  Gfloat    ymax;    /* Y maximum limit */
} Glimit;

typedef union {      /* STRING DATA RECORD */
  Gstringpet0001     stringpet1_datarec;
} Gstringrec;

    typedef struct {
      Gint     bufsiz;         /* buffer size */
      Gint     position;       /* initial cursor position */
      Gchar    *title_string;  /* the title string */
    } Gstringpet0001;
```

## 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

SET STRING MODE
Example 9–3 for a program example using the INITIALIZE STRING function

# INITIALIZE STRING 3

## Operating States

WSOP, WSAC, SGOP

## Syntax

ginitstring3 (

```
  Gint          ws,       /* (I) Workstation identifier. */
  Gint          dev,      /* (I) String device number. */
  Gchar         *init,    /* (I) Initial string. Once you request input,
                               the user can delete or edit the initial
                               string; otherwise, the newly input
                               string is appended to the initial
                               string. */
  Gint          pet,      /* (I) String prompt and echo type. */
  Glimit3       *volume,  /* (I) Echo volume. */
  Gstringrec    *data     /* (I) String data record. */
```

)

## Data Structures

```
typedef struct {            /* COORDINATE LIMITS */
  Gfloat      xmin;         /* X minimum limit */
  Gfloat      xmax;         /* X maximum limit */
  Gfloat      ymin;         /* Y minimum limit */
  Gfloat      ymax;         /* Y maximum limit */
  Gfloat      zmin;         /* Z minimum limit */
  Gfloat      zmax;         /* Z maximum limit */
} Glimit3;

typedef union {             /* STRING DATA RECORD */
  Gstringpet0001           stringpet1_datarec;
} Gstringrec;

    typedef struct {
      Gint     bufsiz;         /* buffer size */
      Gint     position;       /* initial cursor position */
      Gchar    *title_string;  /* the title string */
    } Gstringpet0001;
```

## 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

SET STRING MODE
Example 9–3 for a program example using the INITIALIZE STRING function

# INITIALIZE STROKE

## Operating States

WSOP, WSAC, SGOP

## Syntax

ginitstroke (

```
  Gint        ws,         /* (I) Workstation identifier */
  Gint        dev,        /* (I) Stroke device number */
  Gstroke     *init,      /* (I) Transformation, number of points, and
                             points */
  Gint        pet,        /* (I) Prompt and echo type */
  Glimit      *area,      /* (I) Echo area */
  Gstrokerec  *data       /* (I) Stroke data record */
```

)

## Data Structures

```
typedef struct {         /* STROKE DATA */
  Gint      transform;   /* normalization transformation number */
  Gint      n_points;    /* number of points in stroke */
  Gpoint    *points;     /* points in stroke */
} Gstroke;

    typedef struct {     /* COORDINATE POINT */
      Gfloat    x;       /* X coordinate */
      Gfloat    y;       /* Y coordinate */
    } Gpoint;

typedef struct {      /* COORDINATE LIMITS */
  Gfloat    xmin;     /* X minimum limit */
  Gfloat    xmax;     /* X maximum limit */
  Gfloat    ymin;     /* Y minimum limit */
  Gfloat    ymax;     /* Y maximum limit */
} Glimit;

typedef union {       /* STROKE DATA RECORD */
  Gstrokepet0001      strokepet1_datarec;
  Gstrokepet0002      strokepet2_datarec;
  Gstrokepet0003      strokepet3_datarec;
  Gstrokepet0004      strokepet4_datarec;
} Gstrokerec;

    typedef struct {
      Gint      bufsiz;      /* input buffer size */
      Gint      editpos;     /* editing position */
      Gpoint    interval;    /* X, Y interval */
      Gfloat    time;        /* time interval */
      Gchar     *data;       /* device/implementation dependent data */
    } Gstrokepet0001;

    typedef Gstrokepet0001 Gstrokepet0002;
```

```
typedef struct {
  Gint      bufsiz;      /* input buffer size */
  Gint      editpos;     /* editing position */
  Gpoint    interval;    /* X, Y interval */
  Gfloat    time;        /* time interval */
  Gacf      acf;         /* attribute control flag (constant) */
  Gmkattr   mk;          /* marker attributes */
  Gchar     *data;       /* device/implementation dependent data */
} Gstrokepet0003;

    typedef struct {      /* POLYMARKER ATTRIBUTES */
      Gasf      type;     /* marker type ASF (constant) */
      Gasf      size;     /* marker width ASF */
      Gasf      colour;   /* marker color ASF */
      Gint      mark;     /* marker index */
      Gmkbundl  bundl;    /* marker bundle */
    } Gmkattr;

        typedef struct {     /* POLYMARKER BUNDLE */
          Gint      type;    /* marker type (constant) */
          Gfloat    size;    /* marker size scale factor */
          Gint      colour;  /* polymarker color index */
        } Gmkbundl;

typedef struct {
  Gint      bufsiz;     /* input buffer size */
  Gint      editpos;    /* editing position */
  Gpoint    interval;   /* X, Y interval */
  Gfloat    time;       /* time interval */
  Gacf      acf;        /* attribute control flag */
  Glnattr   ln;         /* line attributes */
  Gchar     *data;      /* device/implementation dependent data */
} Gstrokepet0004;

    typedef struct {      /* POLYLINE ATTRIBUTES */
      Gasf      type;     /* line type ASF */
      Gasf      width;    /* line width ASF */
      Gasf      colour;   /* line color ASF */
      Gint      line;     /* line index */
      Glnbundl  bundl;    /* line bundle */
    } Glnattr;

        typedef struct {     /* POLYLINE BUNDLE */
          Gint      type;    /* line type (constant) */
          Gfloat    width;   /* line width scale factor */
          Gint      colour;  /* polyline color index */
        } Glnbundl;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gacf | GCURRENT | Input data record current values |
| | GSPECIFIED | Input data record specified values |
| Gasf | GBUNDLED | Bundled attributes |
| | GINDIVIDUAL | Individual attributes |
| Marker types | GMK_POINT | Dot |
| | GMK_PLUS | Plus sign |
| | GMK_STAR | Asterisk |
| | GMK_O | Circle |
| | GMK_X | Diagonal cross |

| | | |
|---|---|---|
| Line types | GLN_SOLID | Solid line |
| | GLN_DASHED | Dashed line |
| | GLN_DOTTED | Dotted line |
| | GLN_DASHDOT | Dashed-dotted line |

## 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

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

ginitstroke3 (

```
  Gint        ws,         /* (I) Workstation identifier */
  Gint        dev,        /* (I) Stroke device number */
  Gstroke3    *init,      /* (I) Transformation, number of points, and
                             points */
  Gint        pet,        /* (I) Prompt and echo type */
  Glimit3     *volume,    /* (I) Echo volume */
  Gstrokerec  *data       /* (I) Stroke data record */
```

)

## Data Structures

```
typedef struct {         /* STROKE 3 DATA */
  Gint      transform;   /* normalization transformation number */
  Gint      view;        /* view index used in transformation */
  Gint      n_points;    /* number of points in stroke */
  Gpoint3   *points;     /* points in stroke */
} Gstroke3;

    typedef struct {     /* COORDINATE POINT */
      Gfloat    x;       /* X coordinate */
      Gfloat    y;       /* Y coordinate */
      Gfloat    z;       /* Z coordinate */
    } Gpoint3;

typedef struct {         /* COORDINATE LIMITS */
  Gfloat    xmin;        /* X minimum limit */
  Gfloat    xmax;        /* X maximum limit */
  Gfloat    ymin;        /* Y minimum limit */
  Gfloat    ymax;        /* Y maximum limit */
  Gfloat    zmin;        /* Z minimum limit */
  Gfloat    zmax;        /* Z maximum limit */
} Glimit3;

typedef union {     /* STROKE DATA RECORD */
  Gstrokepet0001    strokepet1_datarec;
  Gstrokepet0002    strokepet2_datarec;
  Gstrokepet0003    strokepet3_datarec;
  Gstrokepet0004    strokepet4_datarec;
} Gstrokerec;

    typedef struct {
      Gint      bufsiz;      /* input buffer size */
      Gint      editpos;     /* editing position */
      Gpoint    interval;    /* X, Y interval */
      Gfloat    time;        /* time interval */
      Gchar     *data;       /* device/implementation dependent data */
    } Gstrokepet0001;

        typedef struct {    /* COORDINATE POINT */
          Gfloat    x;      /* X coordinate */
          Gfloat    y;      /* Y coordinate */
        } Gpoint;
```

```
typedef Gstrokepet0001 Gstrokepet0002;

typedef struct {
  Gint      bufsiz;      /* input buffer size */
  Gint      editpos;     /* editing position */
  Gpoint    interval;    /* X, Y interval */
  Gfloat    time;        /* time interval */
  Gacf      acf;         /* attribute control flag (constant) */
  Gmkattr   mk;          /* marker attributes */
  Gchar     *data;       /* device/implementation dependent data */
} Gstrokepet0003;

    typedef struct {      /* POLYMARKER ATTRIBUTES */
      Gasf      type;     /* marker type ASF */
      Gasf      size;     /* marker width ASF */
      Gasf      colour;   /* marker color ASF */
      Gint      mark;     /* marker index */
      Gmkbundl  bundl;    /* marker bundle */
    } Gmkattr;

        typedef struct {      /* POLYMARKER BUNDLE */
          Gint      type;     /* marker type (constant) */
          Gfloat    size;     /* marker size scale factor */
          Gint      colour;   /* polymarker color index */
        } Gmkbundl;

typedef struct {
  Gint      bufsiz;      /* input buffer size */
  Gint      editpos;     /* editing position */
  Gpoint    interval;    /* X, Y interval */
  Gfloat    time;        /* time interval */
  Gacf      acf;         /* attribute control flag */
  Glnattr   ln;          /* line attributes */
  Gchar     *data;       /* device/implementation dependent data */
} Gstrokepet0004;

    typedef struct {       /* POLYLINE ATTRIBUTES */
      Gasf      type;      /* line type ASF */
      Gasf      width;     /* line width ASF */
      Gasf      colour;    /* line color ASF */
      Gint      line;      /* line index */
      Glnbundl  bundl;     /* line bundle */
    } Glnattr;

        typedef struct {     /* POLYLINE BUNDLE */
          Gint      type;    /* line type (constant) */
          Gfloat    width;   /* line width scale factor */
          Gint      colour;  /* polyline color index */
        } Glnbundl;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gacf | GCURRENT | Input data record current values |
| | GSPECIFIED | Input data record specified values |
| Gasf | GBUNDLED | Bundled attributes |
| | GINDIVIDUAL | Individual attributes |
| Marker types | GMK_POINT | Dot |
| | GMK_PLUS | Plus sign |
| | GMK_STAR | Asterisk |
| | GMK_O | Circle |
| | GMK_X | Diagonal cross |

| | | |
|---|---|---|
| Line types | GLN_SOLID | Solid line |
| | GLN_DASHED | Dashed line |
| | GLN_DOTTED | Dotted line |
| | GLN_DASHDOT | Dashed-dotted line |

## 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

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

ginitval (

```
  Gint          ws,          /* (I) Workstation identifier */
  Gint          dev,         /* (I) Valuator device number */
  Gfloat        init,        /* (I) Initial value */
  Gint          pet,         /* (I) Prompt and echo type */
  Glimit        *area,       /* (I) Echo area */
  Gvalrec       *data        /* (I) Valuator data record */
```

)

## Data Structures

```
typedef struct {     /* COORDINATE LIMITS */
  Gfloat    xmin;    /* X minimum limit */
  Gfloat    xmax;    /* X maximum limit */
  Gfloat    ymin;    /* Y minimum limit */
  Gfloat    ymax;    /* Y maximum limit */
} Glimit;

typedef union {      /* VALUATOR DATA RECORD */
  Gvalpet_0001       valpet_1_datarec;
  Gvalpet_0002       valpet_2_datarec;
  Gvalpet_0003       valpet_3_datarec;
  Gvalpet0001        valpet1_datarec;
  Gvalpet0002        valpet2_datarec;
  Gvalpet0003        valpet3_datarec;
} Gvalrec;

    typedef Gvalpetneg0001 Gvalpet_0001;

        typedef Gvalpet0001 Gvalpetneg0001;

            typedef struct {
              Gfloat  low;            /* low range limit */
              Gfloat  high;           /* high range limit */
              Gchar   *title_string;  /* the title string */
            } Gvalpet0001;

    typedef Gvalpetneg0002 Gvalpet_0002;

        typedef Gvalpetneg0001 Gvalpetneg0002;

    typedef Gvalpetneg0003 Gvalpet_0003;

        typedef Gvalpetneg0001 Gvalpetneg0003;

    typedef Gvalpet0001 Gvalpet0002;

    typedef Gvalpet0001 Gvalpet0003;
```

## INITIALIZE VALUATOR

### 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

SET VALUATOR MODE
Example 9–4 for a program example using the INITIALIZE VALUATOR function

# INITIALIZE VALUATOR 3

## Operating States

WSOP, WSAC, SGOP

## Syntax

ginitval3 (

```
  Gint        ws,         /* (I) Workstation identifier */
  Gint        dev,        /* (I) Valuator device number */
  Gfloat      init,       /* (I) Initial value */
  Gint        pet,        /* (I) Prompt and echo type */
  Glimit3     *volume,    /* (I) Echo volume */
  Gvalrec     *data       /* (I) Valuator data record */
```

)

## Data Structures

```
typedef struct {      /* COORDINATE LIMITS */
  Gfloat    xmin;     /* X minimum limit */
  Gfloat    xmax;     /* X maximum limit */
  Gfloat    ymin;     /* Y minimum limit */
  Gfloat    ymax;     /* Y maximum limit */
  Gfloat    zmin;     /* Z minimum limit */
  Gfloat    zmax;     /* Z maximum limit */
} Glimit3;

typedef union {      /* VALUATOR DATA RECORD */
  Gvalpet_0001       valpet_1_datarec;
  Gvalpet_0002       valpet_2_datarec;
  Gvalpet_0003       valpet_3_datarec;
  Gvalpet0001        valpet1_datarec;
  Gvalpet0002        valpet2_datarec;
  Gvalpet0003        valpet3_datarec;
} Gvalrec;

    typedef Gvalpetneg0001 Gvalpet_0001;

        typedef Gvalpet0001 Gvalpetneg0001;

            typedef struct {
              Gfloat  low;            /* low range limit */
              Gfloat  high;           /* high range limit */
              Gchar   *title_string;  /* the title string */
            } Gvalpet0001;

    typedef Gvalpetneg0002 Gvalpet_0002;

        typedef Gvalpetneg0001 Gvalpetneg0002;

    typedef Gvalpetneg0003 Gvalpet_0003;

        typedef Gvalpetneg0001 Gvalpetneg0003;

    typedef Gvalpet0001 Gvalpet0002;

    typedef Gvalpet0001 Gvalpet0003;
```

## INITIALIZE VALUATOR 3

### 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

SET VALUATOR MODE
Example 9–4 for a program example using the INITIALIZE VALUATOR function

# REQUEST CHOICE

## Operating States

WSOP, WSAC, SGOP

## Syntax

greqchoice (

```
  Gint        ws,             /* (I) Workstation identifier */
  Gint        dev,            /* (I) Stroke device number */
  Gqchoice    *response       /* (O) Status and choice number */
```

)

## Data Structures

```
typedef struct {        /* REQUEST CHOICE */
  Gcstat    status;     /* request status (constant) */
  Gint      choice;     /* choice data */
} Gqchoice;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcstat | GC_OK | Input obtained |
| | GC_NOCHOICE | Triggered without choosing |
| | GC_NONE | No input obtained |

## 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

greqloc (

```
  Gint      ws,             /* (I) Workstation identifier */
  Gint      dev,            /* (I) Locator device number */
  Gqloc     *response       /* (O) Status, transformation, and
                                   location */
```

)

## Data Structures

```
typedef struct {       /* REQUEST LOCATOR */
  Gistat    status;    /* request status (constant) */
  Gloc      loc;       /* locator data */
} Gqloc;

    typedef struct {          /* LOCATOR DATA */
      Gint      transform;    /* normalization transformation number */
      Gpoint    position;     /* locator position */
    } Gloc;

        typedef struct {      /* COORDINATE POINT */
          Gfloat    x;        /* X coordinate */
          Gfloat    y;        /* Y coordinate */
        } Gpoint;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gistat | GOK | Input obtained |
| | GNONE | No 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

greqloc3 (

```
  Gint      ws,          /* (I) Workstation identifier */
  Gint      dev,         /* (I) Locator device number */
  Gqloc3    *response    /* (O) Status, transformation, index, and
                               location */
```

)

## Data Structures

```
typedef struct {         /* REQUEST LOCATOR 3 DATA */
  Gistat    status;      /* request status (constant) */
  Gloc3     loc;         /* request locator data */
} Gqloc3;
```

```
    typedef struct {          /* LOCATOR 3 DATA */
      Gint      transform;    /* normalization transformation number */
      Gint      view;         /* view index */
      Gpoint3   position;     /* locator position */
    } Gloc3;
```

```
        typedef struct {  /* COORDINATE POINT */
          Gfloat    x;    /* X coordinate */
          Gfloat    y;    /* Y coordinate */
          Gfloat    z;    /* Z coordinate */
        } Gpoint3;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gistat | GOK | Input obtained |
| | GNONE | No 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

greqpick (

```
  Gint       ws,             /* (I) Workstation identifier */
  Gint       dev,            /* (I) Stroke device number */
  Gqpick     *response       /* (O) Status, segment identifier, and
                                    pick identifier */
```

)

## Data Structures

```
typedef struct {      /* REQUEST PICK */
  Gpstat    status;   /* request status (constant) */
  Gint      seg;      /* segment */
  Gint      pickid;   /* pick identifier */
} Gqpick;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gpstat | GP_OK | Input obtained |
| | GP_NOPICK | Triggered without picking |
| | GP_NONE | Break during input |

## 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

greqstring (

```
  Gint        ws,           /* (I) Workstation identifier */
  Gint        dev,          /* (I) Stroke device number */
  Gqstring    *response     /* (O) Status and character string */
```

)

## Data Structures

```
typedef struct {       /* REQUEST STRING */
  Gistat    status;    /* request status (constant) */
  Gchar     *string;   /* string data */
} Gqstring;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gistat | GOK | Input obtained |
| | GNONE | No input obtained |

**Note**

Initialize the *string* field with a string, before the call. The string buffer should be at least one more than the maximum possible string size. It is best to specify the string buffer size as 256 characters.

## 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 1 byte larger than the maximum possible string size.
- 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, and writes the character string to the application's buffer. 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 STROKE

## Operating States

WSOP, WSAC, SGOP

## Syntax

greqstroke (

```
  Gint        ws,           /* (I) Workstation identifier */
  Gint        dev,          /* (I) Stroke device number */
  Gqstroke    *response     /* (O) Status, transformation, and stroke */
```

)

## Data Structures

```
typedef struct {       /* REQUEST STROKE */
  Gistat    status;    /* request status (constant) */
  Gstroke   stroke;    /* stroke data */
} Gqstroke;

    typedef struct {          /* STROKE DATA */
      Gint      transform;    /* normalization transformation number */
      Gint      n_points;     /* number of points in stroke */
      Gpoint    *points;      /* points in stroke */
    } Gstroke;

        typedef struct {  /* COORDINATE POINT */
          Gfloat    x;    /* X coordinate */
          Gfloat    y;    /* Y coordinate */
        } Gpoint;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gistat | GOK | Input obtained |
| | GNONE | No input obtained |

**Note**

The field *n_points* should be initialized with the number representing the maximum number of points that will fit in the points buffer. It will be updated to indicate the total number of points in the stroke.

## 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 total number of points in the stroke, and as many of the stroke points as will fit in the buffer.

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_points* field.
- 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 allocated application buffer. 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

greqstroke3 (

```
  Gint        ws,         /* (I) Workstation identifier */
  Gint        dev,        /* (I) Stroke device number */
  Gqstroke3   *response,  /* (O) Status, transformation, view, and
                             stroke */
```

)

## Data Structures

```
typedef struct {      /* REQUEST STROKE 3 DATA */
  Gistat    status;   /* request stroke status (constant) */
  Gstroke3  stroke;   /* request stroke data */
} Gqstroke3;
```

```
    typedef struct {          /* STROKE 3 DATA */
      Gint      transform;    /* normalization transformation number */
      Gint      view;         /* view index used in transformation */
      Gint      n_points;     /* number of points in stroke */
      Gpoint3   *points;      /* points in stroke */
    } Gstroke3;
```

```
        typedef struct {  /* COORDINATE POINT */
          Gfloat    x;    /* X coordinate */
          Gfloat    y;    /* Y coordinate */
          Gfloat    z;    /* Z coordinate */
        } Gpoint3;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gistat | GOK | Input obtained |
| | GNONE | No input obtained |

**Note**

The field *n_points* should be initialized with the number representing the maximum number of points that will fit in the points buffer. It will be updated to indicate the total number of points in the stroke.

## REQUEST STROKE 3

### 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 total number of points in the stroke, and as many of the stroke points as will fit in the buffer.

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_points* field.
- The logical input device's STROKE buffer, whose size is specified in the call to the INITIALIZE STROKE 3 function.

DEC GKS can return a number of points up to the size of the application's buffer. If the size of the entered stroke is larger than the number of points placed in the application's buffer, all additional data is lost.

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

greqval (

```
  Gint       ws,           /* (I) Workstation identifier */
  Gint       dev,          /* (I) Stroke device number */
  Gqval      *response     /* (O) Status, transformation, and stroke */
```

)

## Data Structures

```
typedef struct {       /* REQUEST VALUATOR */
  Gistat    status;    /* request status (constant) */
  Gfloat    val;       /* valuator data */
} Gqval;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gistat | GOK | Input obtained |
| | GNONE | No 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

gsamplechoice (

```
  Gint        ws,           /* (I) Workstation identifier */
  Gint        dev,          /* (I) Stroke device number */
  Gchoice     *response     /* (O) Status and choice number */
```

)

## Data Structures

```
typedef struct {       /* CHOICE DATA */
  Gcstat    status;    /* choice status (constant) */
  Gint      choice;    /* choice number */
} Gchoice;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gcstat | GC_OK | Input obtained |
| | GC_NOCHOICE | 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

gsampleloc (

```
  Gint     ws,             /* (I) Workstation identifier */
  Gint     dev,            /* (I) Locator device number */
  Gloc     *response       /* (O) Normalization transformation and
                                  location */
```

)

## Data Structures

```
typedef struct {         /* LOCATOR DATA */
  Gint      transform;   /* normalization transformation number */
  Gpoint    position;    /* locator position */
} Gloc;

    typedef struct {     /* COORDINATE POINT */
      Gfloat    x;       /* X coordinate */
      Gfloat    y;       /* Y coordinate */
    } Gpoint;
```

## 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

gsampleloc3 (

```
  Gint      ws,          /* (I) Workstation identifier */
  Gint      dev,         /* (I) Locator device number */
  Gloc3     *response    /* (O) Transformation, view index, and location */
```

)

## Data Structures

```
typedef struct {         /* LOCATOR 3 DATA */
  Gint      transform;   /* normalization transformation number */
  Gint      view;        /* view index */
  Gpoint3   position;    /* locator position */
} Gloc3;
```

```
    typedef struct {    /* COORDINATE POINT */
      Gfloat    x;      /* X coordinate */
      Gfloat    y;      /* Y coordinate */
      Gfloat    z;      /* Z coordinate */
    } Gpoint3;
```

## 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

gsamplepick (

```
  Gint      ws,           /* (I) Workstation identifier */
  Gint      dev,          /* (I) Stroke device number */
  Gpick     *response     /* (O) Status, segment identifier, and
                               pick identifiers */
```

)

## Data Structures

```
typedef struct {       /* PICK DATA */
  Gpstat    status;    /* pick status (constant) */
  Gint      seg;       /* pick segment */
  Gint      pickid;    /* pick identifier */
} Gpick;
```

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gpstat | GP_OK | Input obtained |
| | GP_NOPICK | 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 example using the SAMPLE PICK function

# SAMPLE STRING

## Operating States

WSOP, WSAC, SGOP

## Syntax

gsamplestring (

```
  Gint    ws,          /* (I) Workstation identifier */
  Gint    dev,         /* (I) Stroke device number */
  Gchar   *response    /* (O) Returned character string */
```

)

**Note**

Initialize the *response* argument with a string, before the call. The string buffer you give to this call should be at least one more than the maximum possible string size. It is best to specify the string buffer size as 256 characters.

## 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 1 byte larger than the maximum possible string size.
- 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

gsamplestroke (

```
  Gint       ws,           /* (I) Workstation identifier */
  Gint       dev,          /* (I) Stroke device number */
  Gstroke    *response     /* (O) Status, transformation, and stroke */
```

)

## Data Structures

```
typedef struct {          /* STROKE DATA */
  Gint      transform;    /* normalization transformation number */
  Gint      n_points;     /* number of points in stroke */
  Gpoint    *points;      /* points in stroke */
} Gstroke;
```

```
typedef struct {  /* COORDINATE POINT */
  Gfloat    x;    /* X coordinate */
  Gfloat    y;    /* Y coordinate */
} Gpoint;
```

**Note**

The field *n_points* should be initialized with the number representing the maximum number of points that will fit in the points buffer. It will be updated to indicate the total number of 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_points* field.
- 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.

## See Also

INITIALIZE STROKE
SET STROKE MODE

# SAMPLE STROKE 3

## Operating States

WSOP, WSAC, SGOP

## Syntax

gsamplestroke3 (

```
  Gint        ws,           /* (I) Workstation identifier */
  Gint        dev,          /* (I) Stroke device number */
  Gstroke3    *response     /* (O) Transformation, view index, and
                               stroke */
```

)

## Data Structures

```
typedef struct {          /* STROKE 3 DATA */
  Gint      transform;    /* normalization transformation number */
  Gint      view;         /* view index used in transformation */
  Gint      n_points;     /* number of points in stroke */
  Gpoint3   *points;      /* points in stroke */
} Gstroke3;
```

```
typedef struct { /* COORDINATE POINT */
  Gfloat    x;   /* X coordinate */
  Gfloat    y;   /* Y coordinate */
  Gfloat    z;   /* Z coordinate */
} Gpoint3;
```

**Note**

The field *n_points* should be initialized with the number representing the maximum number of points that will fit in the points buffer. It will be updated to indicate the total number of points returned 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 allocate before the call. You must specify the buffer size in the *n_points* field.
- The logical input device's stroke buffer, whose size is specified in the call to the INITIALIZE STROKE 3 function.

## SAMPLE STROKE 3

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.

## See Also

INITIALIZE STROKE 3
SET STROKE MODE

# SAMPLE VALUATOR

## Operating States

WSOP, WSAC, SGOP

## Syntax

gsampleval (

```
  Gint       ws,            /* (I) Workstation identifier */
  Gint       dev,           /* (I) Stroke device number */
  Gfloat     *response      /* (O) Returned value */
```

)

## 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

gsetchoicemode (

```
  Gint       ws,        /* (I) Workstation identifer */
  Gint       dev,       /* (I) Input device number */
  Gimode     mode,      /* (I) Operating mode (constant) */
  Gesw       echo       /* (I) Echo flag (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gimode | GREQUEST | Request mode. This is the default value. |
| | GSAMPLE | Sample mode. |
| | GEVENT | Event mode. |
| Gesw | GECHO | Echo enabled. This is the default value. |
| | GNOECHO | Echo disabled. |

## 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

gsetlocmode (

```
  Gint        ws,        /* (I) Workstation identifer */
  Gint        dev,       /* (I) Input device number */
  Gimode      mode,      /* (I) Operating mode (constant) */
  Gesw        echo       /* (I) Echo flag (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gimode | GREQUEST | Request mode. This is the default value. |
| | GSAMPLE | Sample mode. |
| | GEVENT | Event mode. |
| Gesw | GECHO | Echo enabled. This is the default value. |
| | GNOECHO | Echo disabled. |

## 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

gsetpickmode (

```
  Gint       ws,        /* (I) Workstation identifer */
  Gint       dev,       /* (I) Input device number */
  Gimode     mode,      /* (I) Operating mode (constant) */
  Gesw       echo       /* (I) Echo flag (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gimode | GREQUEST | Request mode. This is the default value. |
| | GSAMPLE | Sample mode. |
| | GEVENT | Event mode. |
| Gesw | GECHO | Echo enabled. This is the default value. |
| | GNOECHO | Echo disabled. |

## 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

gsetstringmode (

```
  Gint       ws,        /* (I) Workstation identifer */
  Gint       dev,       /* (I) Input device number */
  Gimode     mode,      /* (I) Operating mode (constant) */
  Gesw       echo       /* (I) Echo flag (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gimode | GREQUEST | Request mode. This is the default value. |
| | GSAMPLE | Sample mode. |
| | GEVENT | Event mode. |
| Gesw | GECHO | Echo enabled. This is the default value. |
| | GNOECHO | Echo disabled. |

## 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

gsetstrokemode (

```
  Gint       ws,          /* (I) Workstation identifer */
  Gint       dev,         /* (I) Input device number */
  Gimode     mode,        /* (I) Operating mode (constant) */
  Gesw       echo         /* (I) Echo flag (constant) */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gimode | GREQUEST | Request mode. This is the default value. |
| | GSAMPLE | Sample mode. |
| | GEVENT | Event mode. |
| Gesw | GECHO | Echo enabled. This is the default value. |
| | GNOECHO | Echo disabled. |

## 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

gsetvalmode (

```
  Gint       ws,        /* (I) Workstation identifier */
  Gint       dev,       /* (I) Input device number */
  Gimode     mode,      /* (I) Operating mode (constant) */
  Gesw       echo       /* (I) Echo flag */
```

)

## Constants

| Data Type | Constant | Description |
|---|---|---|
| Gimode | GREQUEST | Request mode. This is the default value. |
| | GSAMPLE | Sample mode. |
| | GEVENT | Event mode. |
| Gesw | GECHO | Echo enabled. This is the default value. |
| | GNOECHO | Echo disabled. |

## 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

## 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**

```
/*
 *  This program initializes and samples locator events.  Some of
 *  the calls it uses include:  INQUIRE LOCATOR STATE,
 *  SET LOCATOR MODE, and INITIALIZE LOCATOR.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

# include <stdio.h>
# include <gks.h>     /* GKS C binding header file */

# define DEV_NUM_1   1
# define LOC_PET     1

main ()
{
     Glimit   area;         /* echo area */
     Gint     bufsize;      /* allocated record buffer size */
     Gconn    conid         = GWC_DEF;
     Glocrec  data;         /* locator data record */
     Gint     dev;          /* device number */
     Gint     error;        /* error indicator */
     Gevent   event;        /* event, workstation class, and device number */
     Gloc     init;         /* initial transformation, view, and location */
     Gint     pet;          /* prompt and echo type */
     Gpoint   position1;    /* WC position of the text string */
     Gpoint   position2;    /* WC position of the text string */
     Gpoint   position3;    /* WC position of the text string */
     Gpoint   position4;    /* WC position of the text string */
     Gpoint   position5;    /* WC position of the text string */
     Gpoint   position6;    /* WC position of the text string */
     Gloc     response;     /* normalization, transformation, and location */
     Glocst   state;        /* returned data structure */
     Gint     state_size;   /* required record buffer size */
     Gfloat   timeout;      /* maximum wait period (in seconds) */
     Gint     ws            = 1;
     Gwstype  wstype        = GWS_DEF;

/*
 *  Open the graphics environment:  open GKS, open the workstation,
 *  activate the workstation, and set the deferral state.
 */

    gopengks (0, 0);
    gopenws (ws, &conid, &wstype);
    gactivatews (ws);
    gsetdeferst (ws, GASAP, GALLOWED);
```

(continued on next page)

**Example 9–1 (Cont.) Using a Locator-Class Logical Input Device in Event Mode**

```
/*
 *  Use INQUIRE LOCATOR DEVICE STATE to initialize the variables
 *  you need to pass to the input functions.
 *
 *  GREALIZED tells the graphics handler to return the input values
 *  as they are implemented.  (Use GSET to return the values the way
 *  the application set them.)
 *
 *  After the function call, bufsize contains the amount of the
 *  buffer filled with the written data record.  If state_size is
 *  larger than bufsize, GKS truncated the data record to fit into
 *  the declared buffer.
 *
 *  state.record is a dummy argument.  The device handler ignores the
 *  data record for all supported locator prompt and echo types.
 *
 *  state.e_area is a structure of real numbers that represent the
 *  rectangular echo area in device coordinates.  The echo area defines
 *  a part of the workstation surface from which GKS accepts input
 *  from the input prompt.
 */

   dev        = DEV_NUM_1;
   bufsize    = sizeof (Glocrec);

   ginqlocst (ws, dev, GREALIZED, bufsize, &state_size,
      &state, &error);

/*  Check to see if the error status is NO_ERROR */

    if (error != 0)
       {
       gmessage (ws, "The error status is not 0");
       goto PROGRAM_END;
       }

/*  Set the initial position of the input prompt. */

    state.loc.position.x = 0.9;
    state.loc.position.y = 0.0;

/*  Initialize the logical input device. */

    area = state.e_area;
    data = state.record;
    init.transform = state.loc.transform;
    init.position = state.loc.position;
    pet  = LOC_PET;

    ginitloc (ws, dev, &init, pet, &area, &data);

/*  Activate the logical input device by placing it in event mode. */

    gsetlocmode (ws, dev, GEVENT, GECHO);
```

(continued on next page)

**Example 9–1 (Cont.) Using a Locator-Class Logical Input Device in Event Mode**

```
/*  Instruct the user. */

    position1.x = 0.05;
    position1.y = 0.95;

    position2.x = 0.05;
    position2.y = 0.90;

    gsetcharheight (0.03);
    gtext (&position1, "Move the input prompt upwards.");
    gtext (&position2, "Trigger until I say when to stop.");

/*
 *  Do until the user moves the input prompt closest to the top
 *  of the workstation surface.
 */

   position3.x = 0.05;
   position3.y = 0.85;

   position4.x = 0.05;
   position4.y = 0.80;

   position5.x = 0.05;
   position5.y = 0.75;

   position6.x = 0.05;
   position6.y = 0.70;

/*
 *  In the while loop, the call to AWAIT EVENT immediately checks
 *  the input queue (as specified by the timeout argument of 0.0).
 *  If the user has not yet entered an event or if the application
 *  has removed all reports generated so far, AWAIT EVENT returns
 *  GNCLASS to its event.class argument.
 *
 *  This program uses only the locator input class to generate
 *  events.  The if statement keeps calling GET LOCATOR as long as
 *  the event.class argument is GLOCATOR.  The program stops calling
 *  GET LOCATOR when the event.class argument becomes GNCLASS.
 */

   timeout = 0.0;

   while (response.position.y < 0.9)
     {
     gawaitevent (timeout, &event);

     if (event.class != GNCLASS)
       ggetloc (&response);

/*  Tease the user as the input prompt gets closer. */

    if ((response.position.y > 0.1) && (response.position.y < 0.5))
      gtext (&position3, "You are still far away.");

    if ((response.position.y > 0.5) && (response.position.y < 0.7))
      gtext (&position4, "You are getting closer.");

    if ((response.position.y > 0.7) && (response.position.y < 0.9))
      gtext (&position5, "You are REALLY close.");
    }
```

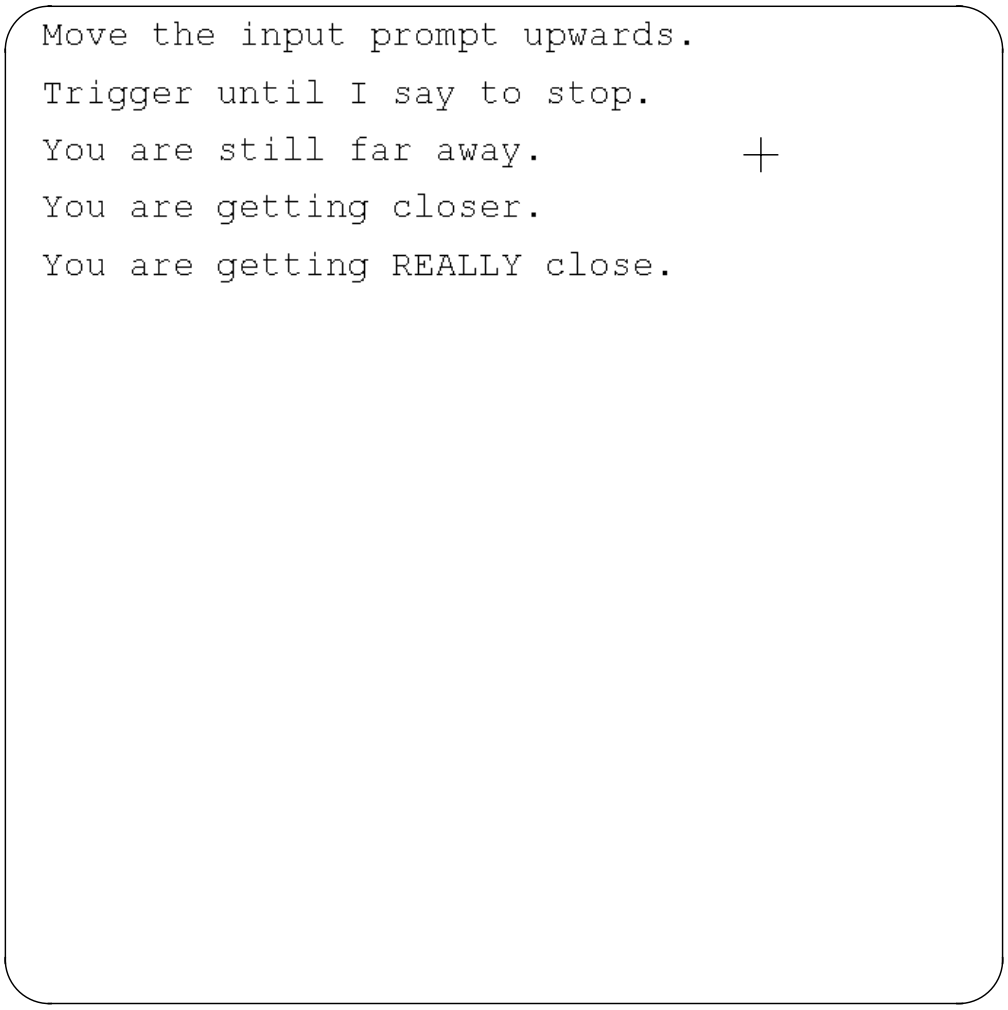
(continued on next page)

**Example 9–1 (Cont.) Using a Locator-Class Logical Input Device in Event Mode**

```
   gtext (&position6, "YOU MADE IT!!!");

/*
 *  Deactivate the logical input device by placing it in request mode.
 *  (You only have to deactivate the device, though, if you are going
 *  to use a different type of input operating mode besides event later
 *  in the program.)
 */

    gsetlocmode (ws, dev, GREQUEST, GECHO);

    PROGRAM_END:

/*
 *  Deactivate the workstation, close the workstation, and
 *  close GKS.
 */

    gdeactivatews (ws);
    gclosews (ws);
    gclosegks ();
}
```

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**

```
/*
 *  This program initializes and samples pick input.  Some of the
 *  calls include:  SET FILL INTERIOR STYLE, SET PICK ID,
 *  SET FILL COLOUR INDEX, FILL AREA, CREATE SEGMENT, CLOSE SEGMENT,
 *  SET TEXT HEIGHT, TEXT, INQUIRE PICK DEVICE STATE, INITIALIZE PICK,
 *  SET PICK MODE, and SAMPLE PICK.
 */

# include <stdio.h>
# include <gks.h>     /* GKS C binding header file */
```

(continued on next page)

**Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode**

```
# define DEV_NUM_1 1
# define LOC_PET   1

main ()
{

/*
 *  state.e_area is an array of real numbers that represent the
 *  rectangular echo area, in device coordinates.  The echo area
 *  defines the workstation surface from which GKS accepts input
 *  from the input prompt.
 */

   Glimit   area;
   Gint     box_1;
   Gint     box_2;
   Gint     bufsize;
   Gint     color_index;
   Gpickrec data;
   Gconn    default_conid;
   Gwstype  default_wstype;
   Gint     dev;
   Gint     error;
   Gpick    init;
   Gint     npoints;
   Gint     pet;
   Gpoint   points[4];
   Gpoint   position1;
   Gpoint   position2;
   Gpoint   position3;
   Gpoint   position4;
   Gpoint   position5;
   Gpoint   position6;
   Gpoint   position7;
   Gpoint   position8;
   Gpoint   position9;
   Gpoint   position10;
   Gpick    response;
   Gpickst  state;
   Gint     state_size;
   Gint     triangle_1;
   Gint     triangle_2;
   Gint     ws_id;

/*
 *  Open the graphics environment:  open GKS, open the workstation,
 *  activate the workstation, and set the deferral state.
 */

    ws_id          = 1;
    default_conid  = GWC_DEF;
    default_wstype = GWS_DEF;

    gopengks (0, 0);
    gopenws (ws_id, &default_conid, &default_wstype);
    gactivatews (ws_id);
    gsetdeferst (ws_id, GASAP, GALLOWED);
```

(continued on next page)

**Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode**

```
/*  Create the divided boxes. */

    gsetfillintstyle (GSOLID);

/*  Establish the position of the first box.
 *
 *  Create the box on the left side of the workstation surface
 *  and place it in a segment.  Divide the box diagonally and
 *  set pick identifiers for each of the created triangles.
 */

    box_1 = 1;
    color_index = 2;
    npoints = 4;
    points[0].x = 0.1;
    points[0].y = 0.3;
    points[1].x = 0.4;
    points[1].y = 0.6;
    points[2].x = 0.1;
    points[2].y = 0.6;
    points[3].x = 0.1;
    points[3].y = 0.3;
    triangle_1 = 1;
    triangle_2 = 2;

    gcreateseg (box_1);
    gsetpickid (triangle_1);
    gsetfillcolourind (color_index);
    gfillarea (npoints, points);

    color_index = 3;
    points[2].x = 0.4;
    points[2].y = 0.3;

    gsetpickid (triangle_2);
    gsetfillcolourind (color_index);
    gfillarea (npoints, points);
    gcloseseg ();

/*  Establish the position of the second box. */

    points[0].x = 0.6;
    points[1].x = 0.9;
    points[2].x = 0.6;
    points[2].y = 0.6;
    points[3].x = 0.6;

/*
 *  Create the box on the right side of the workstation surface
 *  and place it in a segment.  Divide the box diagonally and
 *  set pick identifiers for each of the created triangles.
 */

    gsetpickid (triangle_1);

    box_2 = 2;
    color_index = 2;

    gcreateseg (box_2);
    gsetfillcolourind (color_index);
    gfillarea (npoints, points);

    color_index = 3;
    points[2].x = 0.9;
    points[2].y = 0.3;
```

(continued on next page)

**Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode**

```
    gsetpickid (triangle_2);
    gsetfillcolourind (color_index);
    gfillarea (npoints, points);
    gcloseseg ();

    gsetdet (box_1, GDETECTABLE);
    gsetdet (box_2, GDETECTABLE);

/*  Label the triangles by their pick identifiers. */

    position1.x = 0.20;
    position1.y = 0.45;

    position2.x = 0.30;
    position2.y = 0.45;

    position3.x = 0.70;
    position3.y = 0.45;

    position4.x = 0.80;
    position4.y = 0.45;

    gsetcharheight (0.03);
    gtext (&position1, "1");
    gtext (&position2, "2");
    gtext (&position3, "1");
    gtext (&position4, "2");

/*
 *  Use INQUIRE PICK DEVICE STATE to initialize the variables for
 *  passing the input function.
 *
 *  GREALIZED tells the graphics handler to return the input values
 *  as they are implemented.  (Use GSET to return the values the
 *  way the application set them.)
 */

    dev     = DEV_NUM_1;
    bufsize = sizeof (Gpickst);

    ginqpickst (ws_id, dev, GREALIZED, bufsize, &state_size, &state,
        &error);

/*  Check to see if the error status is NO_ERROR. */

    if (error != 0)
    {
      gmessage (ws_id, "The error status is not 0.");
      goto PROGRAM_END;
    }

/*
 *  Use INITIALIZE PICK to initialize the logical input device.
 *  Assign new values to the input variables.
 */

    area = state.e_area;
    data = state.record;
    init.pickid = state.pick.pickid;
    init.seg    = state.pick.seg;
    init.status = state.pick.status;
```

(continued on next page)

**Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode**

```
    pet = LOC_PET;
    state.pick.pickid = triangle_1;
    state.pick.seg    = box_1;
    state.pick.status = GP_NOPICK;

    ginitpick (ws_id, dev, &init, pet, &area, &data);

/*
 *  Activate the logical input device by placing it in sample mode.
 *  The input prompt now appears on the workstation surface and the
 *  user can change the measure of the device.
 */

    gsetpickmode (ws_id, dev, GSAMPLE, GECHO);

/*  Tell the user the task. */

    position5.x = 0.05;
    position5.y = 0.95;

    position6.x = 0.05;
    position6.y = 0.90;

    gsetcharheight (0.03);
    gtext (&position5, "Move the cursor to a triangle.");
    gtext (&position6, "I will say if it is correct.");

/*
 *  Retrieve the current input value without the user having to
 *  trigger the device by using SAMPLE PICK.  The while loop ends
 *  when the user picks the second triangle in the second box.
 */

    position7.x = 0.05;
    position7.y = 0.85;

    position8.x = 0.05;
    position8.y = 0.80;

    position9.x = 0.05;
    position9.y = 0.75;

    position10.x = 0.05;
    position10.y = 0.70;

    while ((response.seg != 2) || (response.pickid != 2))
       {
       gsamplepick (ws_id, dev, &response);

/*  Tease the user. */

    if ((response.seg == 1) && (response.pickid == 1))
       gtext (&position7, "You are pretty far away.");

    if ((response.seg == 1) && (response.pickid == 2))
       gtext (&position8, "You are getting closer.");
```

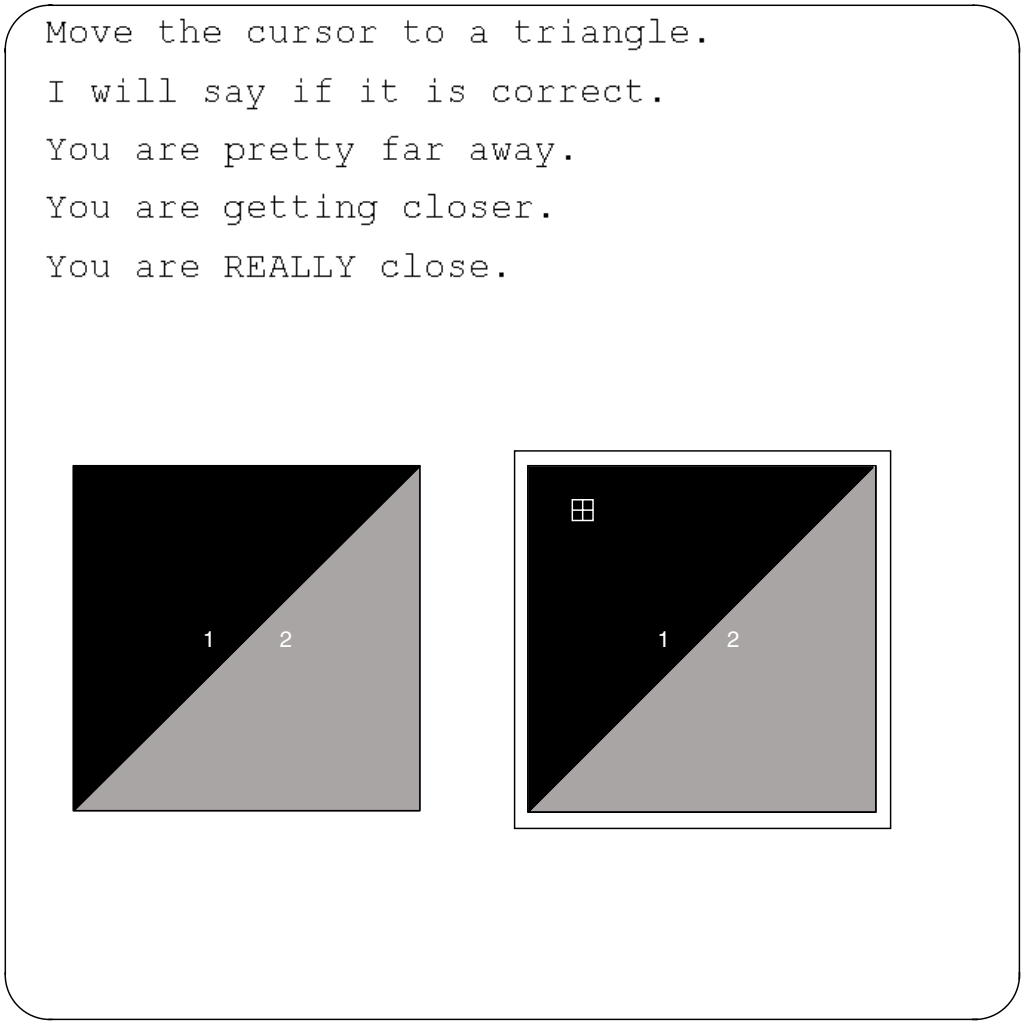
(continued on next page)

**Example 9–2 (Cont.) Using a Pick-Class Logical Input Device in Sample Mode**

```
    if ((response.seg == 2) && (response.pickid == 1))
       gtext (&position9, "You are REALLY close.");
      }

    gtext (&position10, "YOU MADE IT!!!");

/*
 *  Deactivate the logical input device by placing it in request mode.
 *  The device handler removes the input prompt from the workstation
 *  surface and the user can no longer enter input.
 */

    gsetpickmode (ws_id, dev, GREQUEST, GECHO);

    PROGRAM_END:

/*
 *  Deactivate the workstation, close the workstation, and
 *  close GKS.
 */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();
}
```

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**

```
/*
 *  This program initializes and requests string input.  Some of the
 *  calls it uses include:  INQUIRE STRING STATE, INITIALIZE STRING,
 *  SET STRING MODE, and REQUEST STRING.
 *
 *  NOTE: To keep the example concise, no error checking is performed.
 */

# include <stdio.h>
# include <gks.h>      /* GKS C binding definition file */
```

(continued on next page)

**Example 9–3 (Cont.) Using a String-Class Logical Input Device in Request Mode**

```
# define DEV_NUM_1   1
# define LOC_PET     1
# define STRING_SIZE 80

main ()
{

/*
 *  state.record contains the buffer length and the initial
 *  editing position for all logical input prompt and echo types.
 *  The buffer can be only as long as the maximum size the workstation
 *  supports.  To obtain the maximum buffer size, call
 *  INQUIRE DEFAULT STRING DEVICE DATA.
 *
 *  state.e_area is an array of real numbers that represent the
 *  rectangular echo area, in device coordinates.  The echo area
 *  defines the workstation surface from which GKS accepts input
 *  from the input prompt.
 *
 *  The defined string variables contain a string that is the length of
 *  the terminal screen.  You can change the maximum size of the input
 *  string every time you initialize the string logical input device by
 *  changing the value associated with the buffer length.
 */

    Glimit      area;
    Gint        bufsize;
    Gconn       default_conid;
    Gwstype     default_wstype;
    Gint        dev;
    Gint        error;
    Gchar       *init = "GKS>";
    Gint        pet;
    Gqstring    response;
    Gstringst   state;
    Gint        state_size;
    Gchar       string[256];
    Gstringrec  stringrec;
    Gwstype     type;
    Gint        ws_id;


/*
 *  Open the graphics environment:  open GKS, open the workstation,
 *  activate the workstation, and set the deferral state.
 */

  ws_id          = 1;
  default_conid  = GWC_DEF;
  default_wstype = GWS_DEF;

  gopengks (0, 0);
  gopenws (ws_id, &default_conid, &default_wstype);
  gactivatews (ws_id);
  gsetdeferst (ws_id, GASAP, GALLOWED);

/*
 *  Use INQUIRE STRING DEVICE STATE to initialize the variables
 *  you need to pass to the input functions.
 */
```

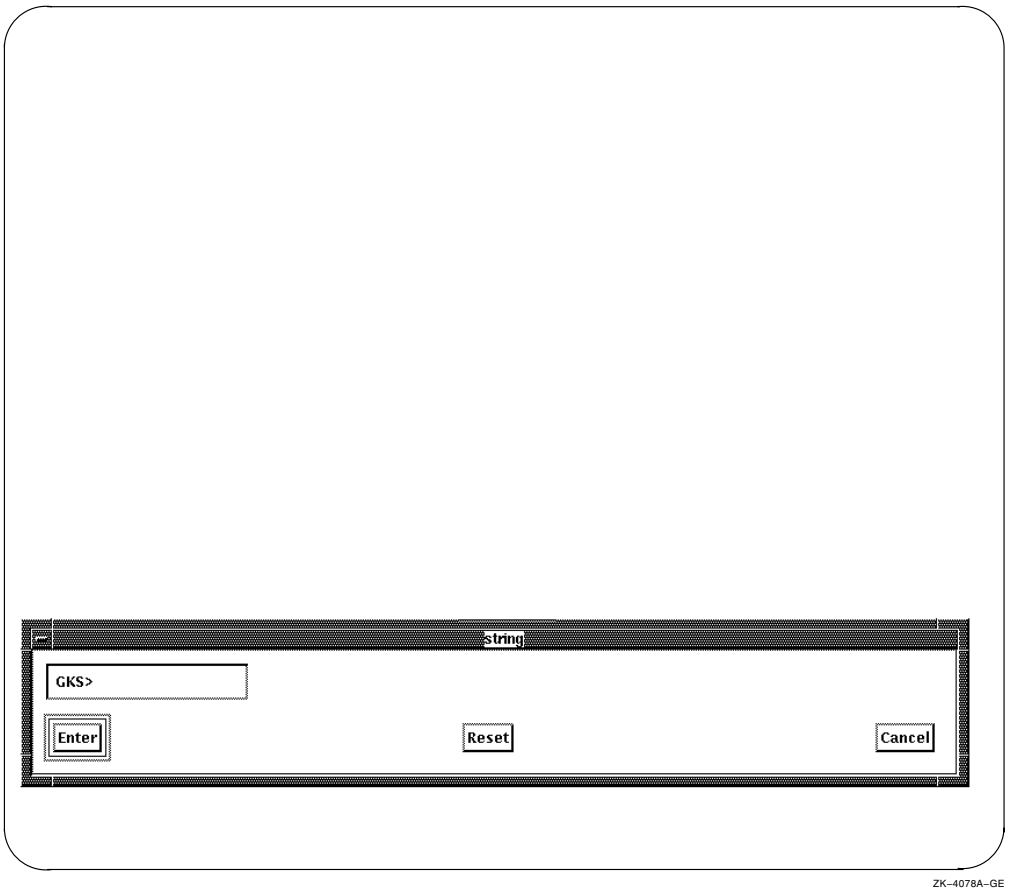
(continued on next page)

**Example 9–3 (Cont.) Using a String-Class Logical Input Device in Request Mode**

```
    bufsize      = sizeof (Gstringst);
    dev          = DEV_NUM_1;
    state.string = string;

    ginqstringst (ws_id, dev, bufsize, &state_size, &state, &error);

/*  Check to see if the error status is NO_ERROR. */

    if (error != 0)
       {
       gmessage (ws_id, "The error status is not 0.");
       goto PROGRAM_END;
       }

/*  Assign new values to the input variables. */

    area = state.e_area;
    pet = LOC_PET;
    stringrec.stringpet1_datarec.position = 1;
    stringrec.stringpet1_datarec.bufsiz = STRING_SIZE;
    stringrec.stringpet1_datarec.data  = state.record.stringpet1_datarec.data;

/*  Initialize the logical input device. */

    ginitstring (ws_id, dev, init, pet, &area, &stringrec);

/*  Activate the logical input device by calling REQUEST STRING. */

    response.string = string;

    greqstring (ws_id, dev, &response);

    PROGRAM_END:

/*
 *  Deactivate the workstation, close the workstation, and
 *  close GKS.
 */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();

/*  Output the input string and its size. */

    printf ("%s \n",response.string);
}
```

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**

```
/*
 *  This program initializes and samples valuator input.  Some of
 *  the calls include:  INQUIRE VALUATOR STATE, INITIALIZE VALUATOR,
 *  SET VALUATOR MODE, and SAMPLE VALUATOR, EVALUATE TRANSFORMATION
 *  MATRIX, and SET SEGMENT TRANSFORMATION.
 */

# include <stdio.h>
# include <gks.h>       /* GKS C binding definition file */

# define DEV_NUM_1 1
# define LOC_PET   1
```

(continued on next page)

**Example 9–4 (Cont.) Using a Valuator-Class Logical Input Device in Sample Mode**

```
main ()
{
/*
 *  state.e_area is an array of real numbers that represent the
 *  rectangular echo area, in device coordinates.  The echo area
 *  defines the workstation surface from which GKS accepts input
 *  from the input prompt.
 *
 *  The graphics handler uses two parts of the valuator input data
 *  record for prompt and echo type 1:  the real value representing
 *  an upper limit and another real value representing a lower limit.
 *
 *  Your terminal might support one of three valuator prompt and echo
 *  types represented by the integers 1, 2, and 3.  Types 1 and 2
 *  prompt the user with a rectangle and a horizontal scale.  To use
 *  the first two types, the user uses the arrow keys or the mouse to
 *  move an arrow along the scale between the upper and lower limits.
 *  With type 3, GKS changes a single digital representation of the real
 *  values between the upper and lower limits.  The user controls the change
 *  of numbers with the arrow keys or the mouse.
 */

    Gfloat    angle;
    Glimit    area;
    Gint      box;
    Gpoint    box_points[5];
    Gchar     buffer[12];
    Gint      bufsize;
    Gconn     default_conid;
    Gwstype   default_wstype;
    Gint      dev;
    Gint      error;
    Gpoint    fixed_point;
    Gfloat    init;
    Gint      pet;
    Gint      points;               /* number of points in the box */
    Gpoint    position1;            /* position of text */
    Gpoint    position2;
    Gfloat    sampled_value;
    Gscale    scale;
    Gvalst    state;
    Gint      state_size;
    Gvalrec   valrec;
    Gpoint    vectors;
    Gint      ws_id;
    Gint      ws_flag;
    Gfloat    xform_matrix[2][3];
```

(continued on next page)

**Example 9–4 (Cont.) Using a Valuator-Class Logical Input Device in Sample Mode**

```
/*
 *  Open the graphics environment:  open GKS, open the workstation,
 *  activate the workstation, and set the deferral state.
 */

    ws_id          = 1;
    default_conid  = GWC_DEF;
    default_wstype = GWS_DEF;

    gopengks (0, 0);
    gopenws (ws_id, &default_conid, &default_wstype);
    gactivatews (ws_id);
    gsetdeferst (ws_id, GASAP, GALLOWED);


/*
 *  Call INQUIRE VALUATOR DEVICE STATE to initialize the variables you
 *  need to pass to the input functions.
 */

    bufsize = sizeof (Gvalst);
    dev     = DEV_NUM_1;

    ginqvalst (ws_id, dev, bufsize, &state_size, &state, &error);

/*  Check to see if the error status is NO_ERROR. */

       if (error != 0)
       {
         gmessage (ws_id, "The error status is not 0.");
         goto PROGRAM_END;
       }

/*  Assign new values to the input variables. */

    area = state.e_area;
    init = 1.0;
    pet  = LOC_PET;
    valrec.valpet1_datarec.low  = 0.001;
    valrec.valpet1_datarec.high = 2.0;
    valrec.valpet1_datarec.data = state.record.valpet1_datarec.data;

/*  Initialize the logical input device. */

    ginitval (ws_id, dev, init, pet, &area, &valrec);

/*
 *  Activate the logical input device by placing it in sample mode.
 *  The input prompt now appears on the workstation surface and
 *  the user can change the measure of the device.
 */

    gsetvalmode (ws_id, dev, GSAMPLE, GECHO);

/*
 *  Create a box.  (The program will scale the box according to
 *  the sample values of the valuator device.)
 */

    box            = 1;
    points         = 5;

    gsetfillintstyle (GSOLID);
    gsetcharheight(0.03);

    box_points[0].x = 0.4;
    box_points[0].y = 0.4;
```

(continued on next page)

**Example 9–4 (Cont.) Using a Valuator-Class Logical Input Device in Sample Mode**

```
    box_points[1].x = 0.6;
    box_points[1].y = 0.4;

    box_points[2].x = 0.6;
    box_points[2].y = 0.6;

    box_points[3].x = 0.4;
    box_points[3].y = 0.6;

    box_points[4].x = 0.4;
    box_points[4].y = 0.4;

    gcreateseg (box);
    gfillarea (points, box_points);
    gcloseseg ();

/*  Display instructions to the user. */

    position1.x = 0.05;
    position1.y = 0.95;

    position2.x = 0.05;
    position2.y = 0.90;

    gtext (&position1, "Change the size of the box.");
    gtext (&position2, "To stop, set the value to 2.0.");

/*  Sample the user input by using SAMPLE VALUATOR */
 *
 * The call to SAMPLE VALUATOR retrieves the current input value
 * (without the user having to trigger the logical input device).
 * The while loop ends when the user moves the prompt to 2.0.
 */

   sampled_value = 1.0;

   while (sampled_value != 2.0)
     {
      gsampleval (ws_id, dev, &sampled_value);

/*
 *  Scale the segment according to the init_value argument
 *  using EVALUATE TRANSFORMATION MATRIX.
 */

    angle         = 0.0;
    fixed_point.x = 0.5;
    fixed_point.y = 0.5;
    scale.x_scale = sampled_value;
    scale.y_scale = sampled_value;
    vectors.x     = 0.0;
    vectors.y     = 0.0;
```

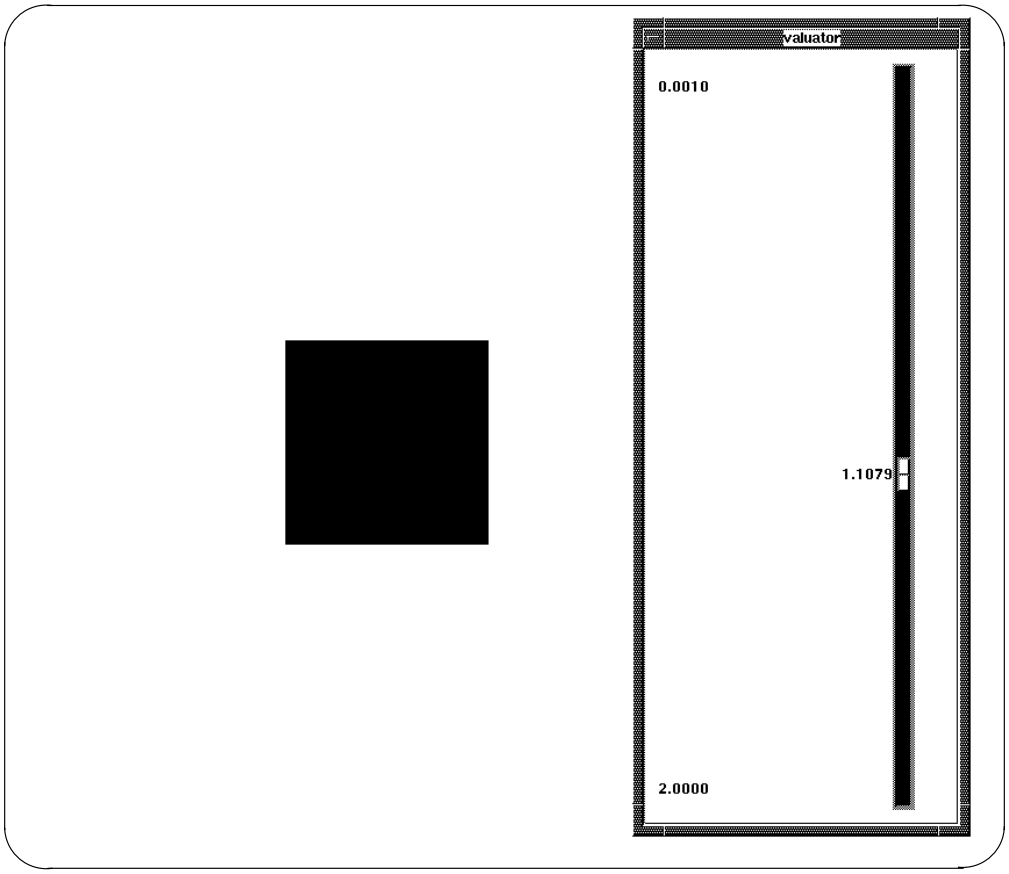
(continued on next page)

**Example 9–4 (Cont.) Using a Valuator-Class Logical Input Device in Sample Mode**

```
    gevaltran (&fixed_point, &vectors, angle, &scale, GWC, xform_matrix);

    if (sampled_value != 1.0)
       {
       gsetsegtran (box, xform_matrix);
       gupdatews (ws_id, GPERFORM);
       }
     }

/*  Deactivate the logical input device by placing it in request mode. */

    gsetvalmode (ws_id, dev, GREQUEST, GECHO);

    PROGRAM_END:

/*  Deactivate the workstation, close the workstation, and close GKS. */

    gdeactivatews (ws_id);
    gclosews (ws_id);
    gclosegks ();

    printf ("%f\n",sampled_value);
}
```

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 GWS_MO (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 GWS_MO workstations as determined by the maximum allowable open and active workstations, sending output to the active GWS_MO 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 GWS_MO 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 GWS_MO workstation is active at the time of the function call. The primitives are stored in a form equivalent to NDC points.
- Output attribute settings that are current at the time of primitive generation.
- Segments, if the GWS_MO 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 GWS_MO workstation and call output functions, DEC GKS formats the graphic output information within the metafile for you.

When you close the GWS_MO workstation, DEC GKS writes an item of type 0 to the metafile to specify that it is the last item in the metafile.

## 10.2 Creating a CGM

To create a CGM, you open and then activate a workstation using the constant GWS_CGMO (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 GWS_CGMO workstations as determined by the maximum allowable open and active workstations, sending appropriate output to the appropriate active type GWS_CGMO workstation.

Once the GWS_CGMO 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 C binding call is as follows:

```
wstype = GWS_CGMO | ECD_MCHAR;

gopenws( wsid, "CGM_METAFILE.TXT", &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 GWS_CGMO 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 GWS_MI (numeric value 3) as the workstation type for category GMI workstations. Also, when you open the type GWS_MI 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 GWS_MI 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 GWS_MI 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 GWS_MI 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 GOUTPUT 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

ggettypegksm (

```
  Gint     ws,         /* (I) Workstation identifier that represents
                              an open metafile input (GWS_MI) workstation. */
  Ggksmit  *result     /* (O) Item type and record length (in bytes).
                              This may be passed to the function READ
                              ITEM FROM GKSM. */
```

)

### Data Structures

```
typedef struct {       /* GKS METAFILE ITEM */
  Gint      type;      /* item type */
  Gint      length;    /* item length */
} Ggksmit;
```

### 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

ginterpret (

```
  Ggksmit    *typeandlength,  /* (I) Item type and record length,
                                    in bytes.  You can obtain this
                                    information by calling the function
                                    GET ITEM TYPE FROM GKSM. */
  Ggksmrec   *data            /* (I) Item's data record. */
```

)

### Data Structures

```
typedef struct {       /* GKS METAFILE ITEM */
  Gint      type;      /* item type */
  Gint      length;    /* item length */
} Ggksmit;

typedef struct {       /* GKS METAFILE DATA RECORD */
  Gchar     *gksmrec;  /* metafile data record */
} Ggksmrec;
```

### 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

greadgksm (

```
  Gint      ws,         /* (I) Integer value that represents an open
                               metafile input (GWS_MI) workstation. */
  Gint      length,     /* (I) Maximum length of data record buffer,
                               in bytes.  If length = 0,
                               DEC GKS ignores the record.  If the
                               actual data record is larger than this
                               maximum value,  DEC GKS truncates the
                               item's data record. */
  Ggksmrec  *record     /* (O) Data record */
```

)

## Data Structures

```
typedef struct {        /* GKS METAFILE DATA RECORD */
  Gchar     *gksmrec;   /* metafile data record */
} Ggksmrec;
```

## 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

gwritegksm (

```
  Gint         ws,          /* (I) Integer value that represents an active
                                   metafile output (GWS_MO) workstation */
  Gint         type,        /* (I) Item type, in the range
                                   type < 0  or type > 100 */
  Gint         length,      /* (I) Length of item's data record,
                                   in bytes */
  Ggksmrec     *data        /* (I) Item's data record */
```

)

### Data Structures

```
typedef struct {       /* GKS METAFILE DATA RECORD */
  Gchar    *gksmrec;   /* metafile data record */
} Ggksmrec;
```

### 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