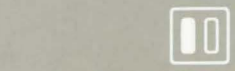


DIGITAL RESEARCH®

GSX™

Graphics Extension

Programmer's Guide

GSX™
Graphics Extension
Programmer's Guide

Digital Research
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950
TWX 910 360 5001

The GSX Graphics Extension Programmer's Guide was prepared using the Digital Research TEX™ Text Formatter and printed in the United States of America.

****
* Second Edition: September 1983 *
****

Foreword

MANUAL OBJECTIVE

This document describes the features and operation of the Graphics System Extension (GSX™), Release 1.2. The manual explains what GSX does and how you can use its graphics capabilities. It also explains how GSX interfaces to your hardware environment and how you can adapt GSX for your own unique graphics devices.

INTENDED AUDIENCE

This manual is intended for microcomputer programmers as well as for system and application programmers who are familiar with operating system and graphics programming concepts.

MANUAL DESIGN

This manual contains five sections, three appendixes, a glossary, and an index. The following descriptions will help you determine a reading path through the manual.

Section 1 is an introduction to GSX. It describes the features you need to know to run graphics application programs.

Section 2 is a programmer's overview of GSX. It explains the GSX architecture and introduces the components of GSX. It also describes how to use GSX with application programs.

Section 3 describes the Graphics Device Operating System (GDOS).

Section 4 describes the Graphics Input/Output System (GIOS). It tells how to interface particular graphics devices to GSX to provide device independence for your application program.

Section 5 provides details about operating GSX and how to integrate your application program with the GSX facilities.

Appendixes contain the following reference information:

Appendix A - GSX conventions for the CP/M® operating system for 8080 microprocessors

Appendix B - GSX conventions for the CP/M-86®, IBM® PC DOS, and MS-DOS™ operating systems for 8086 microprocessors

Appendix C - The Virtual Device Interface (VDI) specification

The glossary follows with terminology unique to GSX. Finally, an extensive index helps you use this document more effectively.

Table of Contents

Table of Contents (continued)

Appendixes

Appendixes (continued)

Appendixes (continued)

Tables and Figures

Tables

Figures

Section 1
INTRODUCTION

ABOUT THIS MANUAL

Section 1 identifies the features of GSX, the Graphics System Extension for your operating system. It explains what GSX does and how to use its graphics functions.

This section is for you if you are a new user of GSX. It assumes that your goal is to quickly hook up your application programs to your system's graphics capability.

If you are a system or an application programmer familiar with operating system concepts, this section introduces you to GSX.

Section 2 through Section 5 provides all the details you need to use GSX with your own unique graphics devices.

GSX BENEFITS

GSX adds graphics to your operating system, as follows:

- GSX supports DR Graph..and DR Draw., two products that extend your graphics capability. DR Graph allows you to graph and plot data by making simple menu selections. DR Draw lets you draw complex graphics images.
- GSX opens a world of application software. You can run any graphics application program that uses GSX with several 8080 and 8086 microcomputer operating systems.
- GSX promotes user portability. The interface between you and GSX is identical to the interface between you and your operating system.
- GSX provides a device-independent software interface for your application programs. You will not need to rewrite your programs if you decide to use a printer instead of a plotter, for example.

GSX FUNCTIONS

All graphics devices are not alike. Terminals, printers, and plotters draw lines, fill in areas, and produce text differently.

With the Graphics System Extension for your operating system, you do not have to worry about device differences, because GSX handles all the differences and lets you talk to the devices through your application program as if the devices were all the same. GSX handles graphics requests and supplies the right program to run the device you are using.

Transforming Points

All computer graphics are displayed on a coordinate system. GSX's job is to make sure the coordinate system that one device uses matches the coordinate system used by another. For example, with GSX your application program produces the same graphics image on your printer that it does on your CRT. The linetypes and character sizes are the same.

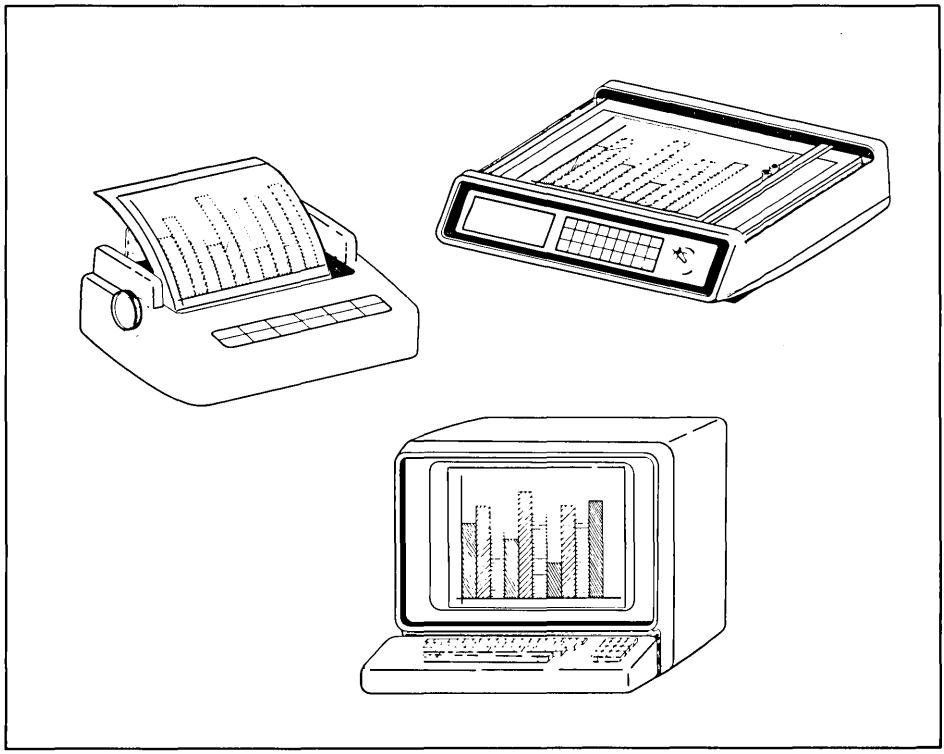


Figure 1-1. GSX Provides Device-Independent Graphics

Servicing Graphics Requests

Your application programs work with GSX through a standard calling sequence. GSX translates these standard calls to fit the peculiarities of each graphics device (a printer or plotter, for example). The translation process makes your application programs device-independent. The programs can run on your system with the graphics device you are using.

For details about using GSX, refer to the GSX user's guide for your system.

Loading Device Drivers

Each graphics device is mechanically and electrically different, and requires a special program to run it. These programs are called device drivers. GSX makes sure the right driver is loaded into memory so you can use the device you specify.

End of Section 1

Section 2
PROGRAMMER'S OVERVIEW

INTRODUCTION

This section introduces the Graphics System Extension architecture with its components and their functions. Later sections describe each of these parts in detail.

GRAPHICS SYSTEM EXTENSION ARCHITECTURE

GSX is the Graphics System Extension for microcomputer operating systems. It incorporates graphics capability into the operating system and provides a host and device-independent interface for your application programs. Graphics primitives are provided for implementing graphics applications with reduced programming effort. In addition, GSX enhances program portability by allowing an application to run on any operating system with the GSX option. GSX also promotes programmer portability by providing a common programming interface to graphics that is compatible with the most widely used operating systems.

GSX is an integral part of your operating system. Application programs interface to GSX through a standard calling sequence. Drivers for specific graphics devices translate the standard GSX calls to the unique characteristics of the device. In this way, GSX provides device independence, and the peculiarities of the graphics device are not visible to the application program.

GSX consists of two parts that work together to give your system graphics capability:

- Graphics Device Operating System (GDOS)
- Graphics Input/Output System (GIOS)

Graphics Device Operating System (GDOS)

The Graphics Device Operating System (GDOS) contains the basic host and device-independent graphics functions that can be called by your application program. GDOS provides a standard interface to graphics that is constant regardless of specific devices or host hardware, just as the disk operating systems standardize disk interfaces. Your application program accesses GDOS in much the same way that it accesses the disk operating system.

GDOS performs coordinate scaling so that your program can specify points in a normalized coordinate space. It uses device-specific information to translate the normalized coordinates into the corresponding values for your particular graphics device.

Multiple graphics devices can be supported under GSX within a single application. By referring to devices with a workstation identification number, an application program can send graphics information to any one of several disk-resident devices. GDOS dynamically loads a specific device driver when requested by the application program, overlaying the previous driver. This technique minimizes memory size requirements since only one driver is resident in memory at any time. For details see "LOADING GIOS FILES" in Section 3.

Graphics Input/Output System (GIOS)

The Graphics Input/Output System (GIOS) is similar to any I/O system. It contains the device-specific code required to interface your particular graphics devices to the GDOS. GIOS consists of a set of device drivers that communicate directly with the graphics devices through the appropriate means. GSX requires a unique device driver for each different graphics device on your system. The term GIOS refers to the functional layer in GSX that holds the collection of available device drivers. The particular driver that is loaded into memory when required by your application is called a GIOS file. Although a single program can use several graphics devices, GDOS loads only one GIOS file at a time.

GIOS performs the graphics primitives of GSX consistent with the inherent capabilities of your graphics device. In some cases, a device driver emulates standard GDOS capabilities that are not provided by the graphics device hardware. For example, some devices require that dashed lines be simulated by a series of short vectors generated in the device driver.

The GSX package contains drivers for many of the most popular graphics devices for microcomputer systems. However, you can install your own custom device driver if necessary. We provide information in Section 4, "GIOS," to help you write your driver. The Virtual Device Interface (VDI) Specification in Appendix C defines all the required functions and parameter conventions.

Enabling Graphics

A special command allows you to enable and disable graphics functions from the command level of the operating system. This command enables GSX by loading GDOS and the default device driver and establishing the proper links to the operating system to allow an application program to access graphics devices. When GSX is disabled, it relinquishes all system memory space, leaving the maximum memory for nongraphics programs.

You must initialize GSX with a graphics command before running an application that uses GSX. Refer to your GSX user's guide for the GSX command that your system uses.

GRAPHICS MODE INITIALIZATION

Upon entering the graphics mode, the operating system performs several actions. First, it brings GDOS into memory along with the default driver, the first device driver listed in the Assignment Table.

Next, it calls the GDOS, which intercepts GDOS calls but passes operating system calls to the operating system.

Finally, control returns to the operating system command interface module, which waits for the next operator command. Note that a warm start (usually invoked by CTRL-Z) does not disturb the graphics mode initialization. However, a cold start, or hardware reboot, disables GSX, which requires you to execute the GSX command after you reboot the system.

Figure 2-1 shows the location of the components of GSX after GSX graphics mode initialization.

When graphics mode is disabled, the memory used by GDOS and the GIOS file is made available to user programs, and control is returned to the operating system user interface module.

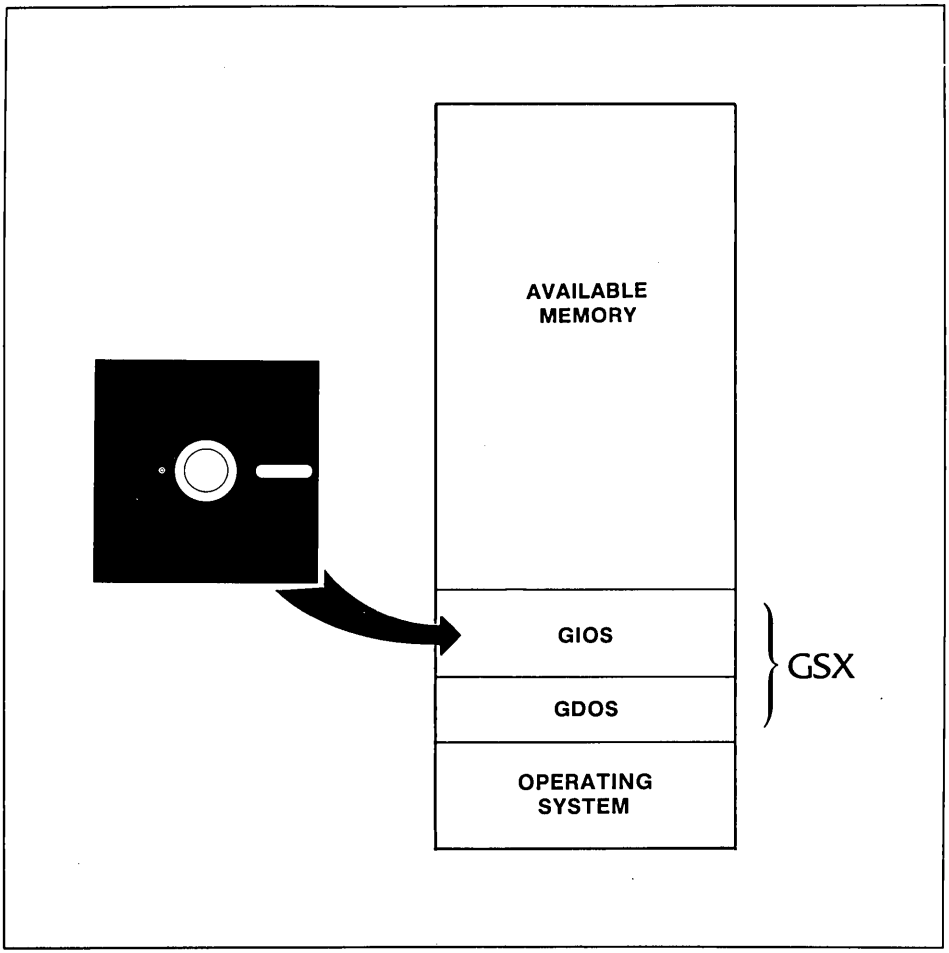


Figure 2-1. GSX Memory Map

APPLICATION PROGRAMS

With appropriate calls to GDOS, you can write your application programs in assembly language or a high-level language that supports the GSX calling conventions. You can compile or assemble and link programs containing GSX calls in the normal manner.

End of Section 2

Section 3
GDOS

INTRODUCTION

This section describes the Graphics Device Operating System (GDOS) in detail, including GDOS functions, the GDOS calling sequence, and how device drivers are loaded.

GDOS FUNCTIONS

GDOS performs three functions during the execution of a graphics application program:

- responds to GSX requests
- loads device drivers as required
- converts normalized coordinates to device coordinates

Graphics Calls

An application program accesses GDOS by making calls to the operating system. Refer to Appendixes A and B for GSX conventions for specific operating systems.

Dynamic Loading

Each time an application program opens a workstation, GDOS determines whether the required device driver is resident in memory. If not, GDOS loads the driver from disk and services the graphics request.

Transforming Points

The application program passes all graphics coordinates to GDOS as Normalized Device Coordinates (NDC) in a range from 0 to 32,767 in both axes. Using information passed from the device driver when the workstation, or device, was opened, GDOS scales the NDC units to the device coordinates. The full scale NDC space is always mapped to the full dimensions of your graphics device in each axis. This ensures that all your graphics information appears on the display surface regardless of the dimensions of the device.

GDOS CALLING SEQUENCE

GSX gives you a standard way to access graphics capabilities. This accessing method is called the Virtual Device Interface (VDI) because it makes all graphics devices appear "virtually" identical.

The implementation of the VDI employs the conventional disk operating system calling sequence. The application program calls GDOS by calling the operating system. For specific operating system calls, refer to Appendixes A and B. The program passes arguments to GDOS in a parameter list, which consists of five arrays: a control array, an array of input parameters, an array of input point coordinates, an array of output parameters, and an array of output point coordinates. The specific graphics function to be performed by GDOS is indicated by an operation code in the parameter list.

GDOS OPCODES

Table 3-1 summarizes the GDOS opcodes. See Appendix C for a detailed description of all the operation codes including parameters.

Table 3-1. GSX Operation Codes

Opcode	Description
1	OPEN WORKSTATION initializes a graphics device (load driver if necessary).
2	CLOSE WORKSTATION stops graphics output to this workstation.
3	CLEAR WORKSTATION clears display device.
4	UPDATE WORKSTATION displays all pending graphics on workstation.
5	ESCAPE enables special device-dependent operation.

ID	Definition
1	INQUIRE ADDRESSABLE CHARACTER CELLS returns number of addressable rows and columns.
2	ENTER GRAPHICS MODE enters graphics mode.
3	EXIT GRAPHICS MODE exits graphics mode.
4	CURSOR UP moves cursor up one row.
5	CURSOR DOWN moves cursor down one row.
6	CURSOR RIGHT moves cursor right one column.
7	CURSOR LEFT moves cursor left one column.
8	HOME CURSOR moves cursor to home position.
9	ERASE TO END OF SCREEN erases from current cursor position to end of screen.
10	ERASE TO END OF LINE erases from current cursor position to end of line.
11	DIRECT CURSOR ADDRESS moves alpha cursor to specified row and column.

Table 3-1. (continued)

Opcode	Description	
	ID	Definition
	12	OUTPUT CURSOR ADDRESSABLE TEXT outputs text at the current alpha cursor position.
	13	REVERSE VIDEO ON displays subsequent text in reverse video.
	14	REVERSE VIDEO OFF displays subsequent text in standard video.
	15	INQUIRE CURRENT CURSOR ADDRESS returns location of alpha cursor.
	16	INQUIRE TABLET STATUS returns status of graphics tablet.
	17	HARDCOPY makes hardcopy.
	18	PLACE GRAPHIC CURSOR AT LOCATION moves cursor directly to specified location.
	19	REMOVE GRAPHIC CURSOR does not display cursor.
	20-50	RESERVED (for future expansion).
	51-100	UNUSED (and available).
6	POLYLINE outputs a polyline.	
7	POLYMARKER outputs markers.	
8	TEXT outputs text starting at specified position.	
9	FILLED AREA displays and fills a polygon.	
10	CELL ARRAY displays a cell array.	

Table 3-1. (continued)

<table>
<tr><th>Opcode</th><th>Description</th></tr>
<tr><td>11</td><td>GENERALIZED DRAWING PRIMITIVE displays a generalized drawing primitive.
<table>
<tr><th>ID</th><th>Definition</th></tr>
<tr><td>1</td><td>BAR</td></tr>
<tr><td>2</td><td>ARC</td></tr>
<tr><td>3</td><td>PIE SLICE</td></tr>
<tr><td>4</td><td>CIRCLE</td></tr>
<tr><td>5</td><td>PRINT GRAPHIC CHARACTERS</td></tr>
<tr><td>6-7</td><td>RESERVED (for future use)</td></tr>
<tr><td>8-10</td><td>UNUSED (and available)</td></tr>
</table></td></tr>
<tr><td>12</td><td>SET CHARACTER HEIGHT sets text size.</td></tr>
<tr><td>13</td><td>SET CHARACTER UP VECTOR sets text direction.</td></tr>
<tr><td>14</td><td>SET COLOR REPRESENTATION defines the color associated with a color index.</td></tr>
<tr><td>15</td><td>SET POLYLINE LINETYPE sets linestyle for polylines.</td></tr>
<tr><td>16</td><td>SET POLYLINE LINEWIDTH sets width of lines.</td></tr>
<tr><td>17</td><td>SET POLYLINE COLOR INDEX sets color for polylines.</td></tr>
<tr><td>18</td><td>SET POLYMARKER TYPE sets marker type for polymarkers.</td></tr>
<tr><td>19</td><td>SET POLYMARKER SCALE sets size for polymarkers.</td></tr>
<tr><td>20</td><td>SET POLYMARKER COLOR INDEX sets color for polymarkers.</td></tr>
<tr><td>21</td><td>SET TEXT FONT sets device-dependent text style.</td></tr>
<tr><td>22</td><td>SET TEXT COLOR INDEX sets color of text.</td></tr>
</table>

Table 3-1. (continued)

Opcode	Description
23	SET FILL INTERIOR STYLE sets interior style for polygon fill (hollow, solid, halftone pattern, hatch).
24	SET FILL STYLE INDEX sets fill style index for polygons.
25	SET FILL COLOR INDEX sets color for polygon fill.
26	INQUIRE COLOR REPRESENTATION returns color representation values of index.
27	INQUIRE CELL ARRAY returns definition of cell array.
28	INPUT LOCATOR returns value of locator.
29	INPUT VALUATOR returns value of valuator.
30	INPUT CHOICE returns value of choice device.
31	INPUT STRING returns character string.
32	SET WRITING MODE sets current writing mode (replace, overstrike, complement, erase).
33	SET INPUT MODE sets input mode (request or sample).

LOADING GIOS FILES The GSX Virtual Device Interface refers to graphics devices as workstations. Before a graphics device can be used, it must first be initialized with an OPEN WORKSTATION operation. This operation initializes the device with selected attributes, such as linetype and color. It also returns information about the device to GDOS.

When the OPEN WORKSTATION operation is performed, GDOS determines whether the correct GIOS file, or device driver, is currently in memory. It does this by comparing the workstation ID specified in the OPEN WORKSTATION call with the workstation ID of the device whose driver is currently loaded. If there is a match (if the correct GIOS file is in memory), the OPEN WORKSTATION request is serviced immediately.

If a match does not occur, the GDOS must load the correct GIOS file. To find it, GDOS refers to a data structure called the Assignment Table, which contains information about the available device drivers and their location.

GDOS searches the Assignment Table for the first device driver entry with a driver number that matches the workstation ID requested in the OPEN WORKSTATION call. If it finds the correct driver entry, GDOS loads the new GIOS file where the previous one was located. When the load is complete, GDOS finishes the OPEN WORKSTATION operation and returns to the calling program.

If there is no match in the Assignment Table when a new driver is required, GDOS returns without loading a driver, and the previous graphics device continues to operate as the open workstation.

Assignment Table Format

The Assignment Table consists entirely of text and can be created or modified with any text editor. It must reside in a file named ASSIGN.SYS on the drive specified in the GSX graphics mode command or on the current default drive if none is specified in the command when GSX is operating. For each device driver, there is an entry containing the driver number, which specifies the workstation ID of the associated device, and the name of the file containing the associated graphics device driver. The name of the device driver file can be any legal unambiguous filename. Any device used during a graphics session must have an entry in the Assignment Table corresponding to the name of its associated driver.

The format for entries in the Assignment Table is as follows:

```
DDXd:filename;comments

DD = logical driver number
X  = space
d  = disk drive code
filename = driver filename (valid unambiguous
           filename of up to eight characters
           and filetype, .SYS extension
           assumed as default)
comments = any text string
```

For example, valid entries in the Table would be as follows:

```
21 A:PRINTR  ; printer
11 A:DDPLOT  ; plotter
 1 B:CRTDRV  ; system console
 2 E:DRIVER.ABC
14 DRIVER2.SYS
```

Note: The driver filename can have any filetype; however, .SYS is assumed if the filetype field is blank. The drive specified in the GSX graphics mode command is used as the default for driver filenames that do not have an explicit drive reference. Extra spaces can be inserted.

The following convention for assigning device driver numbers, or workstation IDs, to graphics devices ensures the maximum degree of device independence within application programs. The convention for driver numbers is as follows:

Device Number	Device Type
1-10	CRT
11-20	Plotter
21-30	Printer
31-40	Metafile
41-50	Other devices

Assign the lowest device number within a device type when you use only one device.

Memory Management

When graphics mode is enabled, GSX allocates memory for the first device driver in the Assignment Table. This driver is referred to as the default device driver. Subsequently, GDOS causes all new drivers to be loaded into the same area where memory was allotted for the original device driver. Ensure that the first driver in the Assignment Table is the largest driver to be loaded so that ample memory space is allocated by the GSX loader for all subsequent drivers. GSX returns an error to the caller and the new driver is not loaded if an attempt is made to load a driver larger than the default driver.

End of Section 3

Section 4
GIOS

INTRODUCTION

This section describes the Graphics Input/Output System, or GIOS. With this information you can write and install your own custom drivers for unique graphic devices.

PURPOSE OF GIOS

As we discussed earlier, GSX is composed of two components: the Graphics Device Operating System (GDOS) and the Graphics Input/Output System (GIOS). GDOS contains the device-independent graphics functions, while GIOS contains the device-dependent code. This division is consistent with the philosophy of isolating device dependencies so that the principal parts of the operating system are transportable to many systems. This also allows applications to run independent of the specific devices connected to the system. In this context, GIOS is analogous to the I/O systems but pertains to graphics devices only. GIOS contains a GIOS file, or device driver, for each of the graphics devices on the system. Each GIOS file contains code to communicate with a single specific graphics device.

A difference between GIOS and I/O systems is that whereas all device drivers contained within I/O systems are resident in memory simultaneously, only one graphics device driver is resident at any time. That is, only one graphics device is active at a time, although the active device can be changed by a request from the application program. GDOS ensures that the correct driver is in memory when required.

GIOS FUNCTIONS

Each of the GIOS files uses the intrinsic graphics capabilities of devices to implement graphics primitives for GDOS. In some cases, the graphics device does not support all the GDOS operations directly, and the driver must emulate the capability in software. For example, if a plotter cannot produce a dashed line, the driver must emulate it by converting a single dashed line into a series of short vectors and transmitting them to the plotter, giving the same end result.

VIRTUAL DEVICE INTERFACE SPECIFICATION

Device drivers must conform to the GSX Virtual Device Interface (VDI) Specification. The VDI specifies the calling sequence to access device driver functions as well as the syntax and semantics of the data structures that communicate across the interface.

The application program passes arguments to device drivers in a parameter list pointed to by the contents of specific registers. The parameter list is in the form of five arrays, as follows:

- control array
- array of input parameters
- array of input point coordinates
- array of output parameters
- array of output point coordinates

The application program specifies the graphics function to be performed by a device driver with an operation code in the control array.

All array elements are type INTEGER (2 bytes). All arrays are 1-based; that is, the double-word address at Parameter Block (PB) points to the first element of the control array (contrl(1)). The meaning of the input and output parameter arrays is dependent on the opcode. See Appendix C, "Virtual Device Interface Specification," for details.

The application program passes all graphics coordinates to the device driver as device coordinates. Using information passed from the device driver when the workstation, or device, was opened, GDOS scales the NDC coordinates, passed from the application to the coordinates of the specific device.

The full-scale NDC space is always mapped to the full dimensions of your graphics device in each axis. This ensures that all your graphics information is visible on the display surface regardless of the actual device dimensions.

However, NDC space is larger than device space. For example, the NDC space for a device is 32K by 32K NDC units. The target device measures 640 by 200 pixels. The size of an NDC pixel is 51 by 164 NDC units. When GSX returns the value of the pixel to an application, the value of the bottom left corner of the NDC pixel is returned by GSX. Therefore, to avoid cumulative errors caused by round-off procedures in your application, you should add an offset of one-half an NDC pixel to the value returned by GSX when you are transforming coordinates up and down GSX.

If your device has an aspect ratio that is not 1:1 (that is, the display surface is not square) and you wish to prevent distortion between your world coordinate system and the device coordinate system, your application must use different scaling factors in the X and Y axes to compensate for the asymmetry of your device. For example, if you are using a typical CRT device with an aspect ratio of 3:4 (vertical:horizontal) to produce a perfect square on the display, you would draw a figure with 4000 NDC units vertically and 3000 NDC units horizontally. That is, the scaling factor for the vertical dimension is 4/3 of the horizontal direction. For most noncritical applications you need not make this adjustment.

Details of the Virtual Device Interface, including required and optional functions and arguments, are included in Appendix C, "Virtual Device Interface Specification."

CREATING A GIOS FILE

Device driver files that are part of GIOS must be in standard executable command format so they can be loaded by GDOS. These files may be renamed to .SYS, the default filetype for GSX GIOS files. You can write a device driver in any language as long as the functions and parameter passing conventions conform to the Virtual Device Interface Specification given above. After assembling or compiling your driver source, link it with any required external subroutines and run-time support libraries to produce a load module.

The name of a GIOS file can consist of eight characters or less with a .SYS filetype. In addition, the driver must be included in the Assignment Table, which is a text file named ASSIGN.SYS on the current default drive.

Refer to "Assignment Table Format" in Section 3 for more details about the ASSIGN.SYS and the correct format for each entry.

End of Section 4

Section 5
OPERATING PROCEDURES

INTRODUCTION

This section explains how to use GSX in your graphics applications.

GSX DISTRIBUTION FILES

When you receive your GSX distribution disk, first check that all required files have been included.

Refer to your GSX user's guide for procedures that check and duplicate the distribution disk.

If any files are missing, contact your distributor to receive a new disk. If all files are present, duplicate the distribution disk using the PIP utility and store your distribution disk in a safe place. Then, using the duplicate disk, transfer the GSX files to a working system disk. Always use the duplicate disk to generate any new copies of GSX. Do not use the distribution disk for routine operations.

RUNNING GRAPHICS APPLICATIONS UNDER GSX

To use the graphics features provided by GSX, you must ensure that several conditions are met:

1. In your application program you must conform to the GSX calling convention to access graphics primitives. This involves making a call to the operating system, which points to a parameter list. This list provides information to GSX and also returns information to the calling program. The details of this procedure are contained in Section 3, "GDOS," Section 4, "GIOS," and the appendixes.

2. Enough stack space must be available for GSX operations. This includes a buffer area for points passed to GSX and some fixed overhead space. The formula to determine the required stack space is discussed below.

3. The required device drivers must be present on the disk specified in the GSX graphics mode command, or in the current default drive if no drive is specified, when your program is executed. Also, the Assignment Table (ASSIGN.SYS) must contain the names of your device drivers and a logical device number or workstation ID that corresponds to the correct device driver. The details of device driver and Assignment Table requirements are included in Section 3, "GDOS," and Section 4, "GIOS."

4. After successfully compiling or assembling and linking your application program you can run it just like any other program, but first you must ensure that GSX is active. You can enable GSX graphics with the GSX graphics mode command documented in the GSX user's guide for your system.

DETERMINING MEMORY REQUIREMENTS

To determine the amount of stack space required to run a given application, make the following calculation:

GSX stack requirements:

Open workstation call = approximately 500 bytes

All others = Ptsin size + 128

Ptsin is the point array passed to the device driver from the application program (two words for each point).

The stack requirement is the largest of the two resulting values. This stack space must be available in the application program stack area.

The memory required by GDOS is less than 3 kilobytes. This is allocated when the GSX graphics mode command is executed. Space for the default device driver is also allocated at this time. The default device driver should be the largest device driver so that sufficient space is allocated for other drivers loaded during execution of your application.

DEBUGGING GRAPHICS APPLICATIONS UNDER GSX

Graphics programs can be debugged with a debugger, as can any GSX application. The default device driver and GDOS are loaded after the command has been executed. Your graphics application program is loaded in the normal manner for applications on your operating system.

WRITING A NEW DEVICE DRIVER

GSX is distributed with a number of device drivers for popular graphics devices. If your devices are included (refer to your GSX user's guide for a summary of the supported devices), you only need to edit the Assignment Table file with a text editor to ensure that it reflects the logical device number assignments that you desire. However, if your device is not supported, you must create a driver program that conforms to the VDI specification. You can write a driver in any language, but at least part of it is usually implemented in assembler due to the low-level hardware interface required.

Your driver must provide the functions listed as required in the VDI specification and must observe the VDI parameter passing conventions. In some cases the capability specified by VDI is not available in the graphics device and the function must be emulated by the driver software. For example, dashed lines can be generated by the driver if they are not directly available in the device. The complete VDI specification is in Appendix C, and the parameter passing conventions are discussed in Section 3, "GDOS," and Section 4, "GIOS."

End of Section 5

Appendix A
GSX CALLING CONVENTIONS FOR CP/M

INTRODUCTION

This appendix briefly outlines the components of a skeleton device driver for GSX on CP/M for 8080 microprocessors. It also summarizes the GSX GDOS calling conventions for CP/M.

GSX SKELETON DEVICE DRIVER

The GSX skeleton device driver describes the components required for a CP/M system.

FORMAT

Function: GSX skeleton device driver

Input Parameters

contrl(1)	--	Opcode for driver function
contrl(2)	--	Number of vertices in array ptsin. Each vertex consists of an x and a y coordinate so the length of this array is twice as long as the number of vertices specified.
contrl(4)	--	Length of integer array intin
contrl(6-n)	--	Opcode dependent information
intin	--	Array of integer input parameters
ptsin	--	Array of input coordinate data

Output Parameters

contrl(3)	--	Number of vertices in array ptsout. Each vertex consists of an x and a y coordinate so the length of this array is twice as long as the number of vertices specified.
contrl(5)	--	Length of integer array intout
contrl(6-n)	--	Opcode dependent information
intout	--	Array of integer output parameters
ptsout	--	Array of output coordinate data

All data passed to the device driver is assumed to be 2-byte INTEGERS.

All coordinates passed to GSX are in Normalized Device Coordinates (0-32767 along each axis). These units are mapped to the actual device units (for example, rasters for CRTs or steps for plotters and printers) by GSX so that all coordinates passed to the device driver are in device units.

Because both input and output coordinates are converted by GSX, both the calling routine and the device driver must ensure that the input vertex count (contrl(2)) and output vertex count (contrl(3)) are set. The calling routine must set contrl(2) to 0 if no x,y coordinates are being passed to GSX. Similarly, the device driver must set contrl(3) to 0 if no x,y coordinates are being returned through GSX.

Because 0-32767 maps to the full extent on each axis, coordinate values are scaled differently on the x and y axes of devices that do not have a square display.

The BDOS call to access GSX and the GIOS in CP/M is as follows:

BDOS opcode (in C register) for GSX call = 115

Parameter Block (address is passed in DE):

```
PB      Address of contrl
PB+1s   Address of intin
PB+2s   Address of ptsin
PB+3s   Address of intout
PB+4s   Address of ptsout
```

s is the number of bytes used for each argument in the parameter block. For CP/M, this is 2 bytes.

All opcodes must be recognized, whether they produce any action or not. A list of required opcodes for CRT devices, plotters, and printers follows the specification. These opcodes must be present and perform as specified. All opcodes should be implemented whenever possible because this gives better quality graphics.

For CP/M, device driver I/O is done through BDOS (Basic Disk Operating System) calls. CRT devices are assumed to be the console device. Plotters are assumed to be connected as the reader or punch device. Printers are assumed to be connected as the list device.

GDOS CALLING CONVENTIONS

The GDOS calling sequence is summarized below.

Function code (in register C) = 115
Parameter block address in register DE

Parameter Block Contents:

PB	Address of control array
PB+2	Address of input parameter array
PB+4	Address of input point coordinate array
PB+6	Address of output parameter array
PB+8	Address of output point coordinate array

Control Array on Input:

contrl(1)	--	Opcode for driver function
contrl(2)	--	Number of vertices in input point array
contrl(4)	--	Length of input parameter array
contrl(6-n)	--	Opcode dependent

Input Parameter Array:

intin	--	Array of input parameters

Input Coordinate Array:

ptsin	--	Array of input coordinates (each point is specified by an X and Y coordinate given in Normalized Device Coordinates between 0 and 32,767)

End of Appendix A

Appendix B
GSX CALLING CONVENTIONS FOR CP/M, IBM PC DOS, AND MS-DOS

INTRODUCTION

This appendix outlines the GSX calling sequence for the GDOS, the procedure for invoking device drivers, and error messages when you use GSX on CP/M-86, IBM PC DOS, and MS-DOS.

GDOS CALLING SEQUENCE

The GDOS calling sequence is outlined below.

Access via interrupt 224

Function code (in register Cx) = 0473h (hex)

Parameter block address in registers Ds-segment and Dx-offset

Parameter Block Contents:

PB	--	Double-word address of control array
PB+4	--	Double-word address of input parameter array
PB+8	--	Double-word address of input point coordinate array
PB+12	--	Double-word address of output parameter array
PB+16	--	Double-word address of output point coordinate array

Control Array on Input:

contrl(1)	--	Opcode for driver function
contrl(2)	--	Number of vertices (not coordinates) in input coordinate point array (ptsin)
contrl(4)	--	Length of input parameter array
contrl(6-n)	--	Opcode dependent (intin)

Input Parameter Array:

intin -- Array of input parameters (length of array is opcode dependent and specified in contrl(4))

Input Point Coordinate Array:

ptsin -- Array of input coordinates (each point is specified by an X and Y coordinate pair given in Normalized Device Coordinates between 0 and 32,767, with length contrl(2)*2)

Control Array on Output:

contrl(3) -- Number of vertices (not coordinates) in output point array (ptsout)
contrl(5) -- Number of elements in output parameter array (intout)
contrl(6-n) -- Opcode dependent

Output Parameter Array:

intout -- Array of output parameters (length of array is opcode dependent)

Output Point Coordinate Array:

ptsout -- Array of output coordinates (each point is specified by an X and Y coordinate pair given in Normalized Device Coordinates between 0 and 32,767) must be greater than the largest possible value of contrl(5)*2.

All array elements are type INTEGER (2 bytes). All arrays are 1-based; that is, the double-word address at PB points to the first element of the control array (contrl(1)). The meaning of the input and output parameter arrays is dependent on the opcode. See Appendix C, "Virtual Device Interface Specification," for details.

GDOS preserves the BP (base pointer) and DS (data segment) registers. All other registers are subject to change when returned from GDOS.

INVOKING DEVICE DRIVERS

Device drivers are invoked with a Callf from GSX and should return with a Retf. The driver must switch to its own stack for internal use, except for an allowed overhead for a few pushes to save the caller's context. The following entry procedure is recommended to provide an error free calling sequence:

```
CGroup          Group   Driver_Code

Driver_Code     CSeg
                Public  Driver

Driver: Mov     Ax,Sp                   ; Save caller's stack pointers
        Mov     Bx,Ss

; Note that Mov Ss,xxx  Mov Sp,xxx is not interruptible on 8086/8088.

        Mov     Ss,StackBase            ; Switch to driver's stack
        Mov     Sp,Offset Top_Stack

        Push    Bx                      ; Push caller's stack pointer
        Push    Ax
        Push    Bp                      ; Save caller's frame
        Push    Ds                      ; Save parameter pointer
        Push    Dx
        Pushf                           ; Save caller's direction flag
```

```
; Invoke the driver.  Ds:Dx points to the parameter block.
;  It returns with a Retf.

        Callf   Dd_Driver               ; Invoke the driver with Ds:Dx

        Popf                            ; Restore caller's direction flag
        Pop     Dx                      ; Restore caller's Ds:Dx
        Pop     Ds
        Pop     Bp                      ; Restore caller's stack frame
        Pop     Ax                      ; Restore caller's Ss:Sp
        Pop     Bx                      ;  via
        Mov     Ss,Bx                   ;  Bx
        Mov     Sp,Ax                   ;  and Ax
        Retf

StackBase       Dw      Seg Top_Stack

Dd_Driver_Code  CSeg
                Extrn   Dd_Driver       :Far

Stack           SSeg
                Rs      16              ; This module pushes 8 words

; Top_Stack is defined in the last module linked in.

                Extrn   Top_Stack       :Byte

                End
```

After coding, assembling and linking your device driver, you have a .CMD file if you use CP/M. First change the filetype to .SYS using the CP/M RENAME command or a similar command for your operating system:

A>REN GIOSXX.SYS=GIOSXX.CMD

Then, to make this driver known to GSX, include its name in the Assignment Table. This table is located in file ASSIGN.SYS and is simply a text file with a specific format containing the names of driver files and the logical device numbers or workstation IDs that you wish to associate with particular devices. Refer to Section 3, "GDOS," or Section 4, "GIOS," for details.

ERROR MESSAGES

In general, registers and flags (including the direction flag) are not restored upon returning from a call to GSX. The GIOS file will preserve the DS, SS and CS registers and BP and SP, but it is not required to preserve any others. GSX does not change any registers as returned from the GIOS except during an OPEN WORKSTATION command. In this case Ax is modified to return status information (the flags are also modified by this command).

The meaning of the contents of Ax on returning from the OPEN WORKSTATION call is as follows:

AL=0	workstation opened successfully
AL=255	error condition--device driver not loaded. In this case AH has a further meaning:

AH	
0	ASSIGN.SYS not found
1	Syntax error in ASSIGN.SYS
2	Device ID not found in ASSIGN.SYS
3	Close error on ASSIGN.SYS
4	Device driver file specified in ASSIGN.SYS not found
5	Device driver file specified in ASSIGN.SYS empty
6	Syntax error on file specified in ASSIGN.SYS (that is, absolute code segment or not .CMD format)
7	Not enough room for file specified

If a read error occurs during the transfer of a GIOS file when an OPEN WORKSTATION call is in progress, the application program is terminated, a message is displayed, and control is returned to the operating system user interface module. The following error messages can be displayed in response to GSX calls:

GSX CS:IP GIOS load error on Id xxxxh (hex)

An error occurred while transferring the device driver from disk. The value of the CS:IP and the device ID are also shown.

GSX CS:IP GIOS invalid

The currently loaded device driver is invalid. This error probably occurred after a load error when the application does not perform an OPEN WORKSTATION command as the first graphics operation.

GSX CS:IP Illegal function: (Cx)

An invalid function code (≠0473h) was specified in Cx. The erroneous code is displayed.

Refer to the GSX user's guide for your system for additional error messages output by GSX.

End of Appendix B

Appendix C
VIRTUAL DEVICE INTERFACE (VDI) SPECIFICATION

INTRODUCTION

This appendix contains the specification of the Virtual Device Interface (VDI). The VDI defines how device drivers interface to GDOS, the device-independent portion of GSX. The context for this document is from the DEVICE DRIVER point of view. All coordinate information is assumed to be in device coordinate space.

FORMAT

Function: GSX graphics operation

Input Parameters

contrl(1)	--	Opcode for driver function.
contrl(2)	--	Number of vertices in array ptsin. Each vertex consists of an x and a y coordinate pair so the length of this array is twice as long as the number of vertices specified.
contrl(4)	--	Length of integer array intin.
contrl(6-n)	--	Opcode dependent information.
intin	--	Array of integer input parameters.
ptsin	--	Array of input point coordinate data.

Output Parameters

contrl(3)	--	Number of vertices in array ptsout. Each vertex consists of an x and a y coordinate pair so the length of this array is twice as long as the number of vertices specified. Other data may be passed back here depending on the opcode.
contrl(5)	--	Length of integer array intout.
contrl(6-n)	--	Opcode dependent information.

intout	--	Array of integer output point parameters.
ptsout	--	Array of output point coordinate data.

Notes

All data passed to the device driver is assumed to be 2-byte INTEGERS, including individual characters in character strings.

All coordinates passed to GSX are in Normalized Device Coordinates (0-32767 along each axis). These units are then mapped to the actual device units (for example, rasters for CRTs or steps for plotters and printers) by GSX so that all coordinates passed to the device driver are in device units.

Because both input and output coordinates are converted by GSX, both the calling routine and the device driver must make sure that the input vertex count (contrl(2)) and output vertex count (contrl(3)) are set. The calling routine must set contrl(2) to 0 if no x,y coordinates are are being passed to GSX. Similarly, the device driver must set contrl(3) to 0 if no x,y coordinates are being returned through GSX. Coordinates returned by GSX are assumed to be the bottom left edge of the pixel. As a consequence, points at the top and right edges of the device coordinate system will not be at the edge of the Normalized Device Coordinates (NDC) system. Exactly how far away they will be is device dependent.

Because 0-32767 maps to the full extent on each axis, coordinate values are scaled differently on the x and y axes of devices that do not have a square display.

All references to arrays are 1-based; that is, subscripted element 1 is the first element in the array.

On calls to the GDOS the number of arguments passed in the intin array (contrl(4)), and the maximum size of the intout array (contrl(5)) should be set by the application. On return to the GDOS by the GIOS the number of arguments in the intout array should be set by the GIOS. Refer to Appendixes A and B for GDOS calling conventions for specific operating systems.

All opcodes must be recognized, whether or not they produce any action. If an opcode is out of range then no action is performed. A list of required opcodes for CRT devices, plotters, and printers follows the specification. These opcodes must be present and perform as specified. All opcodes should be implemented whenever possible since full implementation gives better quality graphics.

Device driver I/O (that is, communication between the device driver and the device via the system hardware ports) is done through operating system calls.

OPEN WORKSTATION Initialize a graphic workstation.

Input

contrl(1)	--	Opcode = 1
contrl(2)	--	0
contrl(4)	--	Length of intin = 10
intin	--	Initial defaults (for example, linestyle color and character size)
intin(1)	--	Workstation identifier (device driver id). This value is used to determine which device driver to dynamically load into memory.
intin(2)	--	Linetype
intin(3)	--	Polyline color index
intin(4)	--	Marker type
intin(5)	--	Polymarker color index
intin(6)	--	Text font
intin(7)	--	Text color index
intin(8)	--	Fill interior style
intin(9)	--	Fill style index
intin(10)	--	Fill color index

Output

contrl(3)	--	Number of output vertices = 6
contrl(5)	--	Length of intout = 45
intout(1)	--	Maximum addressable width of screen/plotter in rasters/steps assuming a 0 start point (for example, a resolution of 640 implies an addressable area of 0-639, so intout(1)=639)
intout(2)	--	Maximum addressable height of screen/plotter in rasters/steps assuming a 0 start point (for example, a resolution of 480 implies an addressable area of 0-479, so intout(2)=479)
intout(3)	--	Device Coordinate units flag 0 = Device capable of producing precisely scaled image (typically plotters and printers) 1 = Device not capable of precisely scaled image (CRTs)
intout(4)	--	Width of one pixel (plotter step, or aspect ratio for CRT) in micrometers
intout(5)	--	Height of one pixel (plotter step, or aspect ratio for CRT) in micrometers

intout(6) -- Number of character heights

0 = continuous scaling

intout(7) -- Number of linetypes
intout(8) -- Number of line widths
intout(9) -- Number of marker types
intout(10) -- Number of marker sizes
intout(11) -- Number of fonts
intout(12) -- Number of patterns
intout(13) -- Number of hatch styles
intout(14) -- Number of predefined colors (must be at least 2 even for monochrome device). This is the number of colors that can be displayed on the device simultaneously.
intout(15) -- Number of Generalized Drawing Primitives (GDPs)
intout(16)-
intout(25) -- Linear list of GDP numbers supported -1 no more GDPs in list. Application should search list until finding a -1 for the desired GDP.

1 -- bar
2 -- arc
3 -- pie slice
4 -- circle
5 -- ruling chars

intout(26)-
intout(35) -- Linear list of attribute set associated with each GDP

-1 -- no more GDPs
0 -- polyline
1 -- polymarker
2 -- text
3 -- fill area
4 -- none

intout(36) -- Color capability flag

0 -- no
1 -- yes

intout(37) -- Text rotation capability flag

0 -- no
1 -- yes

intout(38) -- Fill area capability flag

0 -- no
1 -- yes

intout(39) -- Read cell array operation capability flag

0 -- no
1 -- yes

intout(40) -- Number of available colors (total number of colors in color palette)

0 -- continuous device (more than 32767 colors)
2 -- monochrome (black and white)
>2 -- number of colors available

intout(41) -- Number of locator devices available
intout(42) -- Number of valuator devices available
intout(43) -- Number of choice devices available
intout(44) -- Number of string devices available
intout(45) -- Workstation type

0 -- Output only
1 -- Input only
2 -- Input/Output
3 -- Device independent segment storage
4 -- GKS Metafile output

ptsout(1) -- 0
ptsout(2) -- Minimum character height in device units (not cell size)
ptsout(3) -- 0
ptsout(4) -- Maximum character height in device units (not cell size)
ptsout(5) -- Minimum line width in device units
ptsout(6) -- 0
ptsout(7) -- Maximum line width in device units
ptsout(8) -- 0
ptsout(9) -- 0
ptsout(10) -- Minimum marker height in device units (not cell size)

ptsout(11) -- 0
ptsout(12) -- Maximum marker height in device units (not cell size)

The default color table should be set up differently for a monochrome and a color device.

Monochrome CRT type devices

Index	Color
0	Black
1	White

Monochrome Printer/Plotter devices

Index	Color
0	White
1	Black

Color

Index	Color
0	Black
1	Red
2	Green
3	Blue
4	Cyan
5	Yellow
6	Magenta
7	White
8-n	White

Other default values that should be set by the driver during initialization are as follows:

Character height	= Minimum character height
Character up vector	= 90 degrees counterclockwise from the right horizontal (0 degrees rotation)
Line width	= 1 device unit (raster, plotter step)
Marker height	= Minimum marker height
Writing mode	= Replace
Input mode	= Request for all input classes (locator, valuator, choice, string)

Description

The Open Workstation operation causes a graphics device to become the current device for the application program. The device is initialized with the parameters in the input array and information about the device is returned to GDOS. The graphic device is selected, and, if it is a CRT, the screen is cleared and the alpha device is deselected and blanked.

CLOSE WORKSTATION Stop all graphics output to this workstation.

Input

```
contrl(1) --   Opcode = 2
contrl(2) --   0
```

Output

```
contrl(3) --   0
```

Description

The Close Workstation operation terminates the graphics device properly and prevents any further output to the device. If the device is a CRT, the alpha device is selected, the screen is cleared, and the graphics device is deselected and blanked. If the device is a printer, then an update is executed.

CLEAR WORKSTATION Clear CRT screen or prompt for new paper on plotter.

Input

```
contrl(1) --   Opcode = 3
contrl(2) --   0
```

Output

```
contrl(3) --   0
```

Description

The Clear Workstation operation causes CRT screens to be erased. If the device is a plotter without paper advance, the operator is prompted to load a new page. If the device is a printer a form feed is issued and then an update is executed.

UPDATE WORKSTATION Display all pending graphics on workstation.

Input

```
contrl(1) --   Opcode = 4
contrl(2) --   0
```

Output

```
contrl(3) --   0
```

Description

The Update Workstation operation causes all pending graphics commands that are queued to be executed immediately. The operation is analogous to flushing buffers. For printer drivers this call must be used to start output to the printer.

ESCAPE Perform device specific operation.

Input

```
contrl(1) --   Opcode = 5
contrl(2) --   Number of input vertices
contrl(4) --   Number of input parameters
contrl(6) --   Function identifier

    1       =   INQUIRE ADDRESSABLE CHARACTER
                CELLS
    2       =   ENTER GRAPHICS MODE
    3       =   EXIT GRAPHICS MODE
    4       =   CURSOR UP
    5       =   CURSOR DOWN
    6       =   CURSOR RIGHT
    7       =   CURSOR LEFT
    8       =   HOME CURSOR
    9       =   ERASE TO END OF SCREEN
    10      =   ERASE TO END OF LINE
    11      =   DIRECT CURSOR ADDRESS
    12      =   OUTPUT CURSOR ADDRESSABLE TEXT
    13      =   REVERSE VIDEO ON
    14      =   REVERSE VIDEO OFF
    15      =   INQUIRE CURRENT CURSOR ADDRESS
    16      =   INQUIRE TABLET STATUS
    17      =   HARDCOPY
    18      =   PLACE GRAPHIC CURSOR AT
                LOCATION
    19      =   REMOVE LAST GRAPHIC CURSOR
    20-50   =   UNUSED BUT RESERVED FOR FUTURE
                EXPANSION
    51-100  =   UNUSED AND AVAILABLE FOR USE
```

	intin	--	Function dependent information (described on following pages)
	ptsin	--	Array of input coordinates for escape function
Output	contrl(3)	--	Number of output vertices
	contrl(5)	--	Number of output parameters
	intout	--	Array of output parameters
	ptsout	--	Array of output coordinates

Description

The Escape operation allows the special capabilities of a graphics device to be accessed from the application program. Some escape functions above are predefined, but others can be defined for your particular devices. The parameters passed are dependent on the function being performed.

ESCAPE: INQUIRE ADDRESSABLE CHARACTER CELLS

Return the number of alpha cursor addressable columns and alpha cursor addressable rows.

Input

contrl(2) -- 0
contrl(6) -- Function ID = 1

Output

contrl(3) -- 0
intout(1) -- Number of addressable rows on the screen, typically 24 (-1 indicates cursor addressing not possible)
intout(2) -- Number of addressable columns on the screen, typically 80 (-1 indicates cursor addressing not possible)

Description

This operation returns information to the calling program about the number of vertical (rows) and horizontal (columns) positions where the alpha cursor can be positioned on the screen.

ESCAPE: ENTER GRAPHICS MODE

Enter graphics mode if different from alpha mode.

Input

contrl(2) -- 0
contrl(6) -- Function id = 2

Output

contrl(3) -- 0

Description

This operation causes the graphics device to enter the graphics mode if different than the alpha mode. Used to explicitly exit alpha cursor addressing mode and to transition from alpha to graphic mode properly. The graphics device is selected and cleared. The alpha device is deselected and blanked.

ESCAPE: EXIT GRAPHICS MODE

Exit graphics mode if different from alpha mode.

Input

contrl(2) -- 0
contrl(6) -- Function id = 3

Output

contrl(3) -- 0

Description

The Exit Graphics operation causes the graphics device to exit the graphics mode if different than the alpha mode. Used to explicitly enter the alpha cursor addressing mode and to transition from graphics to alpha mode properly. The alpha device is selected and cleared. The graphics device is deselected and blanked.

ESCAPE: CURSOR UP Move alpha cursor up one row without altering horizontal position.

Input

contrl(2) -- 0
contrl(6) -- Function id = 4

Output

contrl(3) -- 0

Description

This operation moves the alpha cursor up one row without altering the horizontal position. If the cursor is already at the top margin, no action results.

ESCAPE: CURSOR DOWN Move alpha cursor down one row without altering horizontal position.

Input

contrl(2) -- 0
contrl(6) -- Function id = 5

Output

contrl(3) -- 0

Description

This operation moves the alpha cursor down one row without altering the horizontal position. If the cursor is already at the bottom margin, no action results.

ESCAPE: CURSOR RIGHT

Move alpha cursor right one column without altering vertical position.

Input

contrl(2) -- 0
contrl(6) -- Function id = 6

Output

contrl(3) -- 0

Description

The Cursor Right operation moves the alpha cursor right one column without altering the vertical position. If the cursor is already at the right margin, no action results

---

ESCAPE: CURSOR LEFT

Move alpha cursor left one column without altering vertical position.

Input

contrl(2) -- 0
contrl(6) -- Function id = 7

Output

contrl(3) -- 0

Description

The Cursor Left operation causes the alpha cursor to move one column to the left without altering the vertical position. If the cursor is already at the left margin, no action results.

ESCAPE: HOME CURSOR Send cursor to home position.

Input

contrl(2) -- 0
contrl(6) -- Function id = 8

Output

contrl(3) -- 0

Description

This operation causes the alpha cursor to move to the home position, usually the upper left corner of a CRT display.

ESCAPE: ERASE TO END OF SCREEN Erase from current alpha cursor position to the end of the screen.

Input

contrl(2) -- 0
contrl(6) -- Function id = 9

Output

contrl(3) -- 0

Description

This operation erases the display surface from the current alpha cursor position to the end of the screen. The current alpha cursor location does not change.

ESCAPE: ERASE TO END OF LINE

Erase from the current alpha cursor position to the end of the line.

Input

contrl(2) -- 0
contrl(6) -- Function id = 10

Output

contrl(3) -- 0

Description

This operation erases the display surface from the current alpha cursor position to the end of the current line. The current alpha cursor location does not change.

ESCAPE: DIRECT CURSOR ADDRESS

Move alpha cursor to specified row and column.

Input

contrl(2) -- 0
contrl(6) -- Function id = 11
intin(1) -- Row number (1 - number of rows)
intin(2) -- Column number (1 - number of columns)

Output

contrl(3) -- 0

Description

The Direct Cursor Address operation moves the alpha cursor directly to the specified row and column address anywhere on the display surface. Addresses that are beyond the range that can be displayed on the screen are set to the maximum row and/or column accordingly.

ESCAPE: OUTPUT CURSOR ADDRESSABLE TEXT

Output text at the current alpha cursor position.

Input

```
contrl(2) --  0
contrl(4) --  Number of characters in
              character string
contrl(6) --  Function id = 12
intin     --  Text string in ASCII
```

Output

```
contrl(3) --  0
```

Description

This operation displays a string of text starting at the current cursor position. Alpha text characteristics are determined by the attributes currently in effect (for example, reverse video).

ESCAPE: REVERSE VIDEO ON

Display subsequent cursor addressable text in reverse video.

Input

contrl(2) -- 0
contrl(6) -- Function id = 13

Output

contrl(3) -- 0

Description

This operation causes all subsequent text to be displayed in reverse video format; that is, characters are dark on a light background.

ESCAPE: REVERSE VIDEO OFF

Display subsequent cursor addressable text in standard video.

Input

contrl(2) -- 0
contrl(6) -- Function id = 14

Output

contrl(3) -- 0

Description

This operation causes all subsequent text to be displayed in normal video format; that is, characters are light on a dark background.

ESCAPE: INQUIRE CURRENT CURSOR ADDRESS

Return the current cursor position.

Input

contrl(2) -- 0
contrl(6) -- Function id = 15

Output

contrl(3) -- 0
intout(1) -- Row number (1 - number of rows)
intout(2) -- Column number (1 - number of columns

Description

This operation returns the current position of the alpha cursor in row, column coordinates.

ESCAPE: INQUIRE TABLET STATUS

Return tablet status.

Input

contrl(2) -- 0
contrl(6) -- Function id = 16

Output

contrl(3) -- 0
intout(1) -- tablet status

0 = tablet not available
1 = tablet available

Description

This operation returns tablet status whether a graphics tablet, mouse, joystick, or other similar devices are connected to the workstation.

ESCAPE: HARD COPY Generate hardcopy.

Input

contrl(2) -- 0
contrl(6) -- Function id = 17

Output

contrl(3) -- 0

Description

This operation causes the device to generate a hardcopy. This function is very device specific and can entail copying the screen to a printer or other attached hardcopy device.

ESCAPE: PLACE GRAPHIC CURSOR AT LOCATION Place a graphic cursor at specified location

---

Input

contrl(2) -- 2
contrl(6) -- Function id = 18
ptsin(1) -- x-coordinate of location to place cursor
ptsin(2) -- y-coordinate of location to place cursor

Output

contrl(3) -- 0

Description

Place Graphic Cursor at the specified location. This is device dependent and can be an underbar, block, or similar character. This cursor should be the same type as used for request mode locator input. In this way, if sample mode input is supported, the application may use this call to generate the cursor for rubber band type drawing. In memory mapped devices, it is drawn in XOR mode so that it can be removed. The cursor has no attributes; for example, style or color index.

ESCAPE: REMOVE LAST GRAPHIC CURSOR Remove last graphic cursor/marker.

Input

contrl(2) -- 0
contrl(6) -- Function id = 19

Output

contrl(3) -- 0

Description

This operation removes the last graphic cursor placed on the screen.

POLYLINE Output a polyine to device.

Input

```
contrl(1) --   Opcode = 6
contrl(2) --   Number of vertices (x,y pairs)
               in polyline (n)

               ptsin      -- Array of
                             coordinates of
                             polyline in device
                             units (for
                             example, rasters
                             and plotter steps)

               ptsin(1)   -- x-coordinate of
                             first point
               ptsin(2)   -- y-coordinate of
                             first point
               ptsin(3)   -- x-coordinate of
                             second point
               ptsin(4)   -- y-coordinate of
                             second point
                  .
                  .
                  .
               ptsin(2n-1) -- x-coordinate of
                              last point
               ptsin(2n)   -- y-coordinate of
                              last point
```

Output

```
contrl(3) --   0
```

Description

This operation causes a polyline to be displayed on the graphics device. The starting point for the polyline is the first point in the input array. Lines are drawn between subsequent points in the array. Make sure that the lines exhibit the current line attributes: color, linetype, line width. 0 length lines should be displayed. A single coordinate pair should not be displayed.

POLYMARKER Output markers to the device.

Input

```
contrl(1) --   Opcode = 7
contrl(2) --   Number of markers
ptsin     --   Array of coordinates in device
               units (n) (for example, rasters
               and plotter steps)

               ptsin(1) -- x-coordinate of
                           first marker
               ptsin(2) -- y-coordinate of
                           first marker
               ptsin(3) -- x-coordinate of
                           second marker
               ptsin(4) -- y-coordinate of
                           second marker
                  .
                  .
ptsin(2n-1) -- x-coordinate of last marker
ptsin(2n)   -- y-coordinate of last marker
```

Output

```
contrl(3) --   0
```

Description

This operation causes markers to be drawn at the points specified in the input array. Make sure the markers display the current attributes: color, scale, and type.

TEXT Write text at specified position.

Input

contrl(1)	--	Opcode = 8
contrl(2)	--	Number of vertices = 1
contrl(4)	--	Number of characters in text string
intin	--	Word character string in ASCII
ptsin(1)	--	x-coordinate of start point of text in device units
ptsin(2)	--	y-coordinate of start point of text in device units

Output

contrl(3)	--	0

Description

This operation writes text to the display surface starting at the position specified by the input parameters. Note that the X,Y position specified is the lower left corner of the character itself, not the character cell. Also, make sure the text exhibits current text attributes: color, height, character up vector, font. Each word of the intin array contains only one character. Any character code out of range for the selected font should be mapped to a blank.

FILLED AREA Fill a polygon.

Input

```
contrl(1) --   Opcode = 9
contrl(2) --   Number of vertices in polygon
               (n)
ptsin     --   Array of coordinates of polygon
               in device units

ptsin(1) -- x-coordinate of first point
ptsin(2) -- y-coordinate of first point
ptsin(3) -- x-coordinate of second point
ptsin(4) -- y-coordinate of second point


ptsin(2n-1) -- x-coordinate of last point
ptsin(2n)   -- y-coordinate of last point
```

Output

```
contrl(3) --   0
```

Description

This operation fills a polygon specified by the input array with the current fill color. Ensure the correct color, fill interior style (hollow, solid, pattern or hatch) and fill style index are in effect before doing the fill.

If the device cannot do area fill, it must at least outline the polygon in the current fill color. The device driver must ensure that the fill area is closed by connecting the first point to the last point.

A polygon with zero area should be displayed as a dot. A polygon with only one endpoint should not be displayed.

CELL ARRAY Display cell array.

Input

contrl(1)	--	Opcode = 10
contrl(2)	--	2
contrl(4)	--	Length of color index array
contrl(6)	--	Length of each row in color index array (size as declared in a high level language)
contrl(7)	--	Number of elements used in each row of color index array
contrl(8)	--	Number of rows in color index array
contrl(9)	--	Pixel operation to be performed 1 -- replace 2 -- overstrike 3 -- complement (xor) 4 -- erase
intin(1)	--	Color index array (stored one row at time)
ptsin(1)	--	x-coordinate of lower left corner in device units
ptsin(2)	--	y-coordinate of lower left corner in device units
ptsin(3)	--	x-coordinate of upper right corner in device units
ptsin(4)	--	y-coordinate of upper right corner in device units

Output contrl(3) -- 0

Description The Cell Array operation causes the device to draw a rectangular array which is defined by the input parameter X,Y coordinates and the color index array.

The extents of the cell are defined by the lower left-hand and the upper right-hand X,Y coordinates. Within the rectangle defined by those points, the color index array specifies colors for individual components of the cell.

Each row of the color index array should be expanded to fill the entire width of the rectangle specified if necessary, via pixel replication. Each row of the color index array should also be replicated the appropriate number of times to fill the entire height of the rectangular area.

If the device cannot do cell arrays it must at least outline the area in the current line color.

GENERALIZED DRAWING PRIMITIVE (GDP)

Output a primitive display element.

Input

contrl(1) -- Opcode = 11
contrl(2) -- Number of vertices in ptsin
contrl(4) -- Length of input array intin
contrl(6) -- Primitive id

1 -- BAR -- uses fill area attributes (interior style, fill style, fill color)
2 -- ARC -- uses line attributes (color, linetype, width)
3 -- PIE SLICE -- uses fill area attributes (interior style, fill style, fill color)
4 -- CIRCLE -- uses fill area attributes (interior style, fill style, fill color)
5 -- PRINT GRAPHIC CHARACTERS (RULING CHARACTERS)
6 -- 7 are unused but reserved for future expansion
8 -- 10 are unused and available for use

ptsin -- Array of coordinates for GDP

ptsin(1) -- x-coordinate of first point
ptsin(2) -- y-coordinate of first point
ptsin(3) -- x-coordinate of second point
ptsin(4) -- y-coordinate of second point

.
.

ptsin(2n-1) -- x-coordinate of last point
ptsin(2n) -- y-coordinate of last point

intin	--	Data record		
BAR	--	contrl(2)	--	2 (number of vertices
		contrl(6)	--	1 (primitive ID)
		ptsin(1)	--	x-coordinate of lower left-hand corner of bar
		ptsin(2)	--	y-coordinate of lower left-hand corner of bar
		ptsin(3)	--	x-coordinate of upper right-hand corner of bar
		ptsin(4)	--	y-coordinate of upper right-hand corner of bar ARC AND PIE SLICE
		contrl(2)	--	4 (number of vertices)
		contrl(6)	--	2 (ARC) or 3 (PIE SLICE)
		intin(1)	--	Start angle in tenths of degrees (0-3600)
		intin(2)	--	End angle in tenths of degrees (0-3600)
		ptsin(1)	--	x-coordinate of center point of arc
		ptsin(2)	--	y-coordinate of center point of arc
		ptsin(3)	--	x-coordinate of start point of arc on circumference
		ptsin(4)	--	y-coordinate of start point of arc on circumference
		ptsin(5)	--	x-coordinate of end point of arc on circumference

		ptsin(6)	--	y-coordinate of end point of arc on circumference
		ptsin(7)	--	Radius
		ptsin(8)	--	0
	CIRCLE --	contrl(2)	--	3 (number of points)
		contrl(6)	--	4 (primitive id)
		ptsin(1)	--	x-coordinate of center point of circle
		ptsin(2)	--	y-coordinate of center point of circle
		ptsin(3)	--	x-coordinate of point on circumference
		ptsin(4)	--	y-coordinate of point on circumference
		ptsin(5)	--	Radius
		ptsin(6)	--	0
	PRINT GRAPHIC CHARACTERS		--	For graphics on printer (such as Diablo and Epson)
		contrl(2)	--	1 (number of points)
		contrl(4)	--	Number of characters to output
		contrl(6)	--	5
		intin	--	Graphic characters to output
		ptsin(1)	--	x-coordinate of start point of characters
		ptsin(2)	--	y-coordinate of start point of characters
Output	contrl(3) --	0		

Description

The Generalized Drawing Primitive (GDP) operation allows you to take advantage of the intrinsic drawing capabilities of your graphics device. Special elements such as arcs and circles can be accessed through this mechanism. Several primitive identifiers are predefined and others are available for expansion.

The control and data arrays are dependent on the nature of the primitive.

In some GDPs (Arc, Circle, Pie slice) redundant but consistent information is provided. Only the necessary information for a particular device need be used. Also, all angle specifications assume that 0 degrees is 90 degrees to the right of vertical, with values increasing in the counterclockwise direction.

SET CHARACTER HEIGHT Set character height.

Input

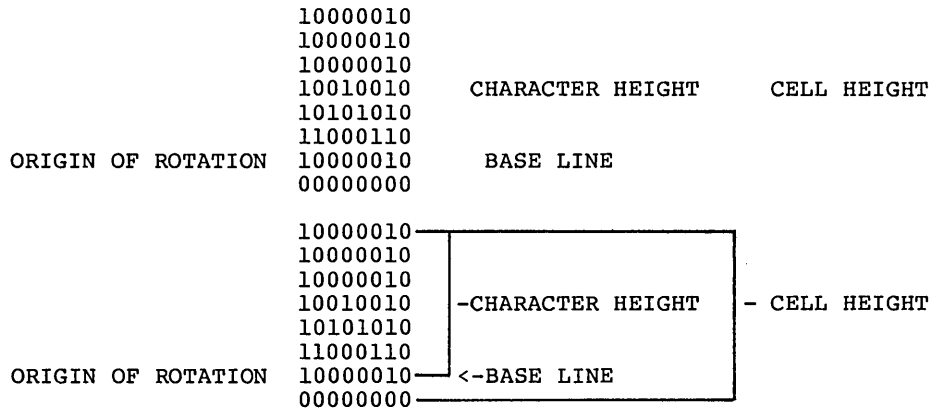
contrl(1) -- Opcode = 12
contrl(2) -- Number of vertices = 1
ptsin(1) -- 0
ptsin(2) -- Requested character height in device units (rasters, plotter steps)

Output

contrl(3) -- Number of vertices = 2
ptsout(1) -- Actual character width selected in device units
ptsout(2) -- Actual character height selected in device units
ptsout(3) -- Character cell width in device units
ptsout(4) -- Character cell height in device units

Description

This operation sets the current text character height in Device Units. The specified height is the height of the character itself rather than the character cell. The driver returns the size of both the character and the character cell. The character size is defined as the size of an uppercase W. If the requested size does not exist, a smaller size should be used.



SET CHARACTER UP VECTOR Set text direction.

Input

contrl(1)	--	Opcode = 13
contrl(2)	--	0
intin(1)	--	Requested angle of rotation of character baseline (in tenths of degrees 0 - 3600)
intin(2)	--	Run of angle = cos (angle) * 100 (0-100)
intin(3)	--	Rise of angle = sin (angle) * 100 (0-100)

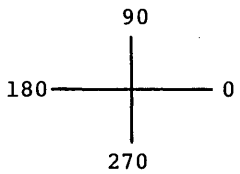
Output

contrl(3)	--	0
contrl(5)	--	1
intout(1)	--	Angle of rotation of character baseline selected (in tenths of degrees 0-3600)

Description

This operation requests an angle of rotation specified in tenths of degrees for the CHARACTER UP VECTOR, which specifies the baseline for subsequent text. The driver returns the actual up direction that is a best fit match to the requested value.

For convenience, redundant but consistent information is provided on input. Only information pertinent to a given device need be used. The angle specification assumes that 0 degrees is 90 degrees to the right of vertical (east on a compass), with angles increasing in the counterclockwise direction.



SET COLOR REPRESENTATION Specify color index value.

Input

```
contrl(1) --   Opcode = 14
contrl(2) --   0
intin(1)  --   Color index
intin(2)  --   Red color intensity      (in
               tenths of percent 0- 1000)
intin(3)  --   Green color intensity
intin(4)  --   Blue color intensity
```

Output

```
contrl(3) --   0
```

Description

This operation associates a color index with the color specified in RGB units. At least two color indexes are required (black and white for monochrome). On a monochrome device, any percentage of color should be mapped to white. On color devices without palettes, a simple remapping of the color indexes is sufficient. On color devices with palettes, loading the palette map is the proper operation. If the color index requested is out of range, no operation is performed.

SET POLYLINE LINETYPE Set polyline linetype.

Input

```
contrl(1) --   Opcode = 15
contrl(2) --   0
intin(1)  --   Requested linestyle
```

Output

```
contrl(3) --   0
intout(1) --   Linestyle selected
```

Description

This operation sets the linetype for subsequent polyline operations. The total number of linestyles available is device dependent; however, 5 linestyles are required: one solid plus four dash styles.

If the requested linestyle is out of range, use linestyle 1 (solid).

```
STYLE -- 1     SOLID        1111111111111111
STYLE -- 2     DASH         1111111000000000
STYLE -- 3     DOT          1110000011100000
STYLE -- 4     DASH,DOT     1111111000111000
STYLE -- 5     LONG DASH    1111111111110000
```

SET POLYLINE LINE WIDTH

Set polyline line width.

Input

contrl(1) -- Opcode = 16
contrl(2) -- Number of input vertices = 1
ptsin(1) -- Requested line width in device units
ptsin(2) -- 0

Output

contrl(3) -- Number of output vertices = 1
ptsout(1) -- Selected line width in device units
ptsout(2) -- 0

Description

This operation sets the width of lines for subsequent polyline operations. Any attempt to set the width beyond the specified maximum will set it to the maximum line width.

SET POLYLINE COLOR INDEX

Set polyline color index.

Input

contrl(1) -- Opcode = 17
contrl(2) -- 0
intin(1) -- Requested color index

Output

contrl(3) -- 0
intout(1) -- Color index selected

Description

This operation sets the color index for subsequent polyline operations. The color signified by the index is determined by the SET_COLOR_REPRESENTATION operation. At least two color indexes are required. Color indexes range from 0 to a device-dependent maximum. If the selected index is out of range, use the MAXIMUM color index.

SET POLYMARKER TYPE Set polymarker type.

Input

contrl(1) -- Opcode = 18
contrl(2) -- 0
intin(1) -- Requested polymarker type

Output

contrl(3) -- 0
intout(1) -- Polymarker type selected

Description

This operation sets the marker type for subsequent polymarker operations. The total number of markers available is device-dependent; however, five marker types are required, as follows:

1 - .	Dot
2 - +	Plus
3 - *	Asterisk
4 - O	Circle
5 - X	Diagonal Cross

If the requested marker type is out of range, use type 3. Marker 1 should always be implemented as the smallest dot that can be displayed.

SET POLYMARKER SCALE Set polymarker scale (height).

Input

contrl(1) -- Opcode = 19
contrl(2) -- Number of input vertices = 1
ptsin(1) -- 0
ptsin(2) -- Requested polymarker height in device units

Output

contrl(3) -- Number of output vertices = 1
ptsout(1) -- 0
ptsout(2) -- Polymarker height selected in device units

Description

This operation requests a polymarker height for subsequent polymarker operations. The driver returns the actual height selected. If the selected height does not exist, use a smaller height.

SET POLYMARKER COLOR INDEX Set polymarker color index.

Input

contrl(1) -- Opcode = 20
contrl(2) -- 0
intin(1) -- Requested polymarker color index

Output

contrl(3) -- 0
intout(1) -- Polymarker color index selected

Description

This operation sets the color index for subsequent polymarker operations. The value of the index is specified by the COLOR operation. At least two color indexes are required. If the index is out of range, use the MAXIMUM color index.

SET TEXT FONT Set the hardware text font.

Input

```
contrl(1) --   Opcode = 21
contrl(2) --   0
intin(1)  --   Requested hardware text font
               number
```

Output

```
contrl(3) --   0
intout(1) --   Hardware text font selected
```

Description

This operation selects a character font for subsequent text operations. Fonts are device-dependent and are specified from 1 to a device-dependent maximum.

SET TEXT COLOR INDEX Set color index.

Input

```
contrl(1) --   Opcode = 22
contrl(2) --   0
intin(1)  --   Requested text color index
```

Output

```
contrl(3) --   0
intout(1) --   Text color index selected
```

Description

This operation sets the color index for subsequent text operations. At least two color indexes are required. Color indexes range from 0 to a device-dependent maximum. If the selected index is out of range, use the MAXIMUM index.

SET FILL INTERIOR STYLE Set interior fill style.

Input

```
contrl(1) --   Opcode = 23
contrl(2) --   0
intin(1)  --   Requested fill interior style

               0 - Hollow (outline no fill)
               1 - Solid
               2 - Halftone pattern
               3 - Hatch
```

Output

```
contrl(3) --   0
intout(1) --   Fill interior style selected
```

Description

This operation sets the fill interior style to be used in subsequent polygon fill operations. If the requested style is not available, use Hollow. The style actually used is returned to the calling program.

SET FILL STYLE INDEX Set fill style index.

Input

```
contrl(1) --   Opcode = 24
contrl(2) --   0
intin(1)  --   Requested fill style index for
               Pattern or Hatch fill
```

Output

```
contrl(3) --   0
intout(1) --   fill style index selected for
               Pattern or Hatch fill
```

Description

Select a fill style based on the fill interior style. This index has no effect if the interior style is either Hollow or Solid. Indexes go from 1 to a device-dependent maximum. If the requested index is not available, use index 1. The index references a hatch style if the fill interior style is hatch, or it references a halftone pattern if the interior fill style is halftone pattern. For consistency, the hatch styles should be implemented in the following order:

```
1  -- vertical lines
2  -- horizontal lines
3  -- +45° lines
4  -- -45° lines
5  -- cross
6  -- X
>6 -- device-dependent
```

You can implement halftone patterns for gray scale shading with values 1 through 6. Value 1 is the lightest, and 6 is the darkest.

SET FILL COLOR INDEX Set fill color index.

Input

```
contrl(1) --   Opcode = 25
contrl(2) --   0
intin(1)  --   Requested fill color index
```

Output

```
contrl(3) --   0
intout(1) --   Fill color index selected
```

Description

This operation sets the color index for subsequent polygon fill operations. The actual RGB value of the color index is determined by the SET-COLOR-REPRESENTATION operation. At least two color indexes are required. Color indexes range from 0 to a device-dependent maximum. If the selected index is out of range, use the MAXIMUM.

INQUIRE COLOR REPRESENTATION Return color representation.

Input

```
contrl(1) --  Opcode = 26
contrl(2) --  0
intin(1)  --  Requested color index
intin(2)  --  Set or realized flag
              0 = set (return color values
                  requested)
              1 = realized (return color
                  values realized on device)
```

Output

```
contrl(3) --  0
intout(1) --  Color index
intout(2) --  Red intensity (in tenths of
              percent 0-1000)
              intout(3) --  Green intensity
              intout(4) --  Blue intensity
```

Description

This operation returns the requested or the actual value of the specified color index in RGB units.

Note: The device driver must maintain tables of the color values that were set (requested) and the color values that were realized. On devices that have a continuous color range, one of these tables may not be necessary. If the selected index is out of range, use the values for the MAXIMUM color index.

INQUIRE CELL ARRAY Return cell array definition.

Input

contrl(1) -- Opcode = 27
contrl(2) -- 2
contrl(4) -- Length of color index array
contrl(6) -- Length of each row in color index array
contrl(7) -- Number of rows in color index array
ptsin(1) -- x-coordinate of lower left corner in device units
ptsin(2) -- y-coordinate of lower left corner in device units
ptsin(3) -- x-coordinate of upper right corner in device units
ptsin(4) -- y-coordinate of upper right corner in device units

Output

contrl(3) -- 0
contrl(8) -- Number of elements used in each row of color index array
contrl(9) -- Number of rows used in color index array
contrl(10) -- Invalid value flag

0 -- If no errors
1 -- If a color value could not be determined for some pixel

intout -- Color index array (stored one row at time)

-1 -- Indicates that a color index could not be determined for that particular pixel

Description

This operation returns the cell array definition of the specified cell. Color indexes are returned one row at a time, starting from the top of the rectangular area, proceeding downward.

INPUT LOCATOR Return locator position.

For REQUEST MODE

Input

contrl(1) -- Opcode = 28
contrl(2) -- Number of input vertices = 1
intin(1) -- Locator device number

1 = keyboard
2 = mouse, joystick

ptsin(1) -- Initial x-coordinate of locator in device units
ptsin(2) -- Initial y-coordinate of locator in device units

Output

contrl(3) -- Number of output vertices = 1
contrl(5) -- Length of intout array -- status

0 = request unsuccessful
>0 = request successful

intout(1) -- Locator terminator
For keyboard terminated locator input, this is the ASCII character code of the key struck to terminate input. For input that is not keyboard-terminated (such as from a tablet or mouse), valid locator terminators begin with <space> (ASCII 32) and increase from there. For instance, if the puck on a tablet has 4 buttons, the first button should generate a <space> as a terminator, the second a <!> (ASCII 33), the third a <"> (ASCII 34), and the fourth a <#> (ASCII 35).
ptsout(1) -- Final x-coordinate of locator in device units
ptsout(2) -- Final y-coordinate of locator in device units

Description for Request Mode

This operation returns the position in Device Coordinates of the specified locator device. Upon entry to the locator routine, a GRAPHIC cursor is placed at the initial coordinate. The GRAPHIC cursor is tracked with the input device until a terminating even occurs, which can result from the user pressing a key, or a button on a mouse. The cursor is removed when the terminating event occurs.

For SAMPLE MODE

Input

contrl(1) -- Opcode = 28
contrl(2) -- Number of input vertices = 1
intin(1) -- Locator device number

1 = keyboard
2 = mouse, joystick

Output

Table C-1. Sample Mode Status Returned

Event	Control Array (3)	(5)
Coordinates Change	1	0
Key Pressed; Coordinates Not Changed	0	1
No Input	0	1

Output

contrl(3) -- Number of output vertices

1 = coordinate changed
0 = no coordinate changed

contrl(5) -- Length of intout array

0 = no terminating character
1 = terminating character returned

intout(1)	--	Locator terminator if terminating event occurs. For keyboard terminated locator input, this is the ASCII character code of the key struck to terminate input in the low byte and 0 in the high byte. For input that is not keyboard-terminated (such as from a tablet or mouse), valid locator terminators begin with 20 hex (ASCII 32) and increase from there.
ptsout	--	Returned if coordinate changed
ptsout(1)	--	New x-coordinate of locator in device units
ptsout(2)	--	New y-coordinate of locator in device units

Description for Sample Mode

Upon entry to the locator routine, NO cursor is displayed. Input is sampled. If the coordinate changed, it is returned and contrl(3) is set to 1. Contrl(5) is set to 0. If a terminating event occurs, a character is returned and contrl(5) is set to 1. Contrl(3) is set to 0. If nothing happens, neither a character nor coordinate is returned.

INPUT VALUATOR Return value of valuator device.

For REQUEST MODE

Input

contrl(1) -- Opcode = 29
contrl(2) -- 0
intin(2) -- Initial value

Output

contrl(3) -- 0
contrl(5) -- 1 length of intout array

intout(1) -- Output value

intout(2) -- Terminator
The terminating character is returned as an ASCII character for keyboard input with the high byte set to 0.

Description for Request Mode

This operation returns the current value of the valuator device. The initial value of the valuator is incremented or decremented (typically with the Up Arrow and Down Arrow keys) until a terminating character is struck.

Typical implementation of the Up Arrow and Down Arrow keys is as follows:

- Pressing the Up Arrow key adds 10 to the valuator.
- Pressing the Down Arrow key subtracts 10 from the valuator.

However, when the Up and Down arrow keys are pressed with the Shift key, the following occurs:

- Up Arrow key adds 1 to the valuator.
- Down Arrow key subtracts 1 from the valuator.

For SAMPLE MODE

Input

contrl(1) -- opcode = 29
contrl(2) -- 0

Output

contrl(3) -- 0
contrl(5) -- Length of intout array
status

0 = nothing happened
1 = valuator changed
2 = terminating character

intout(1) -- New valuator value
intout(2) -- Terminator if terminating event occurred

Description for Sample Mode

This operation returns the current value of the valuator device. The valuator device is sampled. If the valuator changed, the valuator value is incremented or decremented as required. If a terminating event occurred, the value is returned. If nothing happens, no value is returned.

INPUT CHOICE Return choice device status keys.

For REQUEST MODE

Input

```
contrl(1) --  Opcode = 30
contrl(2) --  0
intin(1)  --  Choice device number

              1  = function keys
              >1 = workstation-dependent
```

Output

```
contrl(3) --  0
contrl(5) --  1

intout(1) --  Choice number (range of valid
              numbers beginning at 1 to
              workstation-dependent maximum)
```

Description for Request Mode

This operation returns the choice from the selected choice device. Upon entry to the routine, the keys are sampled until a valid choice key is pressed. This choice is returned. The range for choice numbers begins at 1; its maximum value is device-dependent. Input Choice is typically implemented as function keys.

For SAMPLE MODE

Input

```
contrl(1) --  Opcode = 30
contrl(2) --  0
intin(1)  --  Choice device number

              1  = function keys
              >1 = workstation-dependent
```

Output

```
contrl(3) --  0
contrl(5) --  Choice status

              0 = nothing happened
              1 = sample successful
              2 = nonchoice key

intout(1) --  Choice number if sample
              successful
intout(2) --  Choice terminator if terminating
              event occurs
```

Description for Sample Mode

This operation returns the choice status of the selected choice device. Upon entry to the routine, input is sampled. If input is available and it is a valid choice key, it is returned. If input is available but it is not from a choice key, it is returned as a terminating event. The range of choice numbers begins at 1; its maximum value is device-dependent.

INPUT STRING Return string from specified string device.

For REQUEST MODE

Input

contrl(1)	--	Opcode = 31
contrl(2)	--	0 if nonecho mode 1 if echo mode
intin(1)	--	String device number 1 = default string device (keyboard)
intin(2)	--	Maximum string length
intin(3)	--	Echo mode 0 = do not echo input characters 1 = echo input characters
ptsin(1)	--	x coordinate of echo area in echo mode
ptsin(2)	--	y coordinate of echo area in echo mode

Output

contrl(3)	--	0
contrl(5)	--	1 0 = request unsuccessful >0 = request successful
intout	--	Output string

Description for Request Mode

This operation returns a string from the specified device. Upon entry input is accumulated until a carriage return is encountered or the intout array is full. If echo mode is enabled, text should be echoed to the screen with the current text attributes: color, height, character up vector, and font.

For SAMPLE MODE

Input

contrl(1) -- Opcode = 31
contrl(2) -- 0
intin(1) -- String device number

1 = default string device (keyboard)

intin(2) -- Maximum string length

Output

contrl(3) -- 0
contrl(5) -- Length of output string

0 = sample unsuccessful (characters not available)
>0 = sample successful (characters available)

intout -- Output string if sample successful

Description for Sample Mode

This operation returns a string from the specified device. Upon entry to the routine, input is sampled. If data is available, it is accumulated. Input is sample again. Input is accumulated until one of the following occurs:

- Input is accumulated until it is no longer available
- A carriage return is encountered.
- The intout buffer is full.

Note that sample mode returns immediately as soon as no input is available.

SET WRITING MODE Set writing mode

Input

```
contrl(1) --   Opcode = 32
contrl(2) --   0
intin(1)  --   Writing mode

               1 = replace
               2 = transparent
               3 = XOR (complement)
               4 = erase
```

Output

```
contrl(3) --   0
intout    --   Writing mode selected
```

Description

This operation affects the way pixels from lines, filled areas, and text are placed on the display.

The following are descriptions of the four writing modes used by the GSX:

- MASK is the line style mask.
- FORE is the selected color after mapping from GSX.
- BACK is the color 0 after mapping from GSX (default is black).
- OLD is the current PIXEL color value.
- NEW is the replacement color value.

REPLACE MODE Replace mode is insensitive to the currently displayed image. Any information already displayed is completely replaced. The mask refers to the line style or fill pattern.

Boolean Expression NEW = (FORE and MASK) or (BACK and not MASK)

TRANSPARENT MODE

Transparent mode only affects the pixels where the mask is one and these are changed to the FORE value.

Boolean Expression NEW = (FORE and MASK) or (OLD and not MASK)

XOR MODE XOR mode reverses the bits representing the color.

Boolean Expression NEW = (FORE and MASK) XOR OLD

ERASE MODE Erase mode sets the display to the currently selected background color where the mask value is one, independent of the foreground color.

Boolean Expression (NEW = BACK and MASK) or (OLD and not MASK)

SET INPUT MODE Set input mode.

Input

```
contrl(1) --   Opcode = 33
contrl(2) --   0
intin(1)  --   Logical input device

               1 = locator
               2 = valuator
               3 = choice
               4 = string

intin(2)  --   Input mode

               1 = request
               2 = sample
```

Output

```
contrl(3) --   0
intout(1) --   Input mode selected
```

Description

This operation sets the input mode for the specified logical input device (locator, valuator, choice, string) to either request or sample. In request mode, the driver waits until an input event occurs before returning. In sample mode, the driver returns the current status or location of the input device without waiting.

REQUIRED OPCODE FOR CRT DEVICES

The following opcodes and subfunctions are required for CRT devices:

Table C-2. Opcode for CRT Devices

Opcode	Description
1	Open workstation
2	Close workstation
3	Clear workstation
4	Update workstation
5	Escape
	Id / Definition
	1 Inquire addressable character cells 2 Exit graphics mode 3 Enter graphics mode 4 Cursor up 5 Cursor down 6 Cursor right 7 Cursor left 8 Home cursor 9 Erase to end of screen 10 Erase to end of line 11 Direct cursor address 12 Output cursor addressable text 15 Inquire current cursor address 18 Place graphic cursor 19 Remove graphic cursor
6	Polyline
7	Polymarker
8	Text
9	Filled area
10	Cell array
11	Graphic Drawing Primitive (GDP)
	Id / Definition
	1 Bar Fill

Table C-2. (continued)

Opcode	Description
12	Set character height
14	Set color representation
15	Set polyline linetype
17	Set polyline color index
18	Set polymarker type
20	Set polymarker color index
22	Set text color index
25	Set fill color index
26	Inquire color representation
33	Set input mode (required only if input locator, input valuator, input choice, or input string is present)

REQUIRED OPCODE FOR PLOTTERS AND PRINTERS

The following opcodes and subfunctions are required for plotters and printers:

Table C-2. Opcode for CRT Devices

Opcode	Definition	
1	Open workstation	
2	Close workstation	
3	Clear workstation	
4	Update workstation	
5	Escape	
	Id	Definition
	1	Inquire addressable character cells
6	Polyline	
7	Polymarker	
8	Text	
9	Filled area	
10	Cell array	
11	Graphic Drawing Primitive (GDP)	
	Id	Definition
	1	Bar Fill
12	Set character height	
14	Set color representation	

Table C-2. (continued)

Opcode	Description
15	Set polyline linetype
17	Set polyline color index
18	Set polymarker type
20	Set polymarker color index
22	Set text color index
25	Set fill color index
26	Inquire color representation
33	Set input mode (required only if input locator, input valuator, input choice, or input string is present)

Determining if an opcode that is not required is available in a particular driver can be done in a couple of ways. One way is to check the information about available features returned from the OPEN WORKSTATION opcode. Another way is to check the selected value returned from an opcode against the requested value. If the two values do not match, then either the opcode was not available or the requested value was not available, and a best fit value was selected.

End of Appendix C

Glossary

assignment table Associates logical device numbers, called workstation IDs, with specific GIOS files so that devices can be referred to by number within the application program. The Assignment Table resides in a text file called ASSIGN.SYS and can be modified using any text editor.

BDOS Basic Disk Operating System for the CP/M family of operating systems. It contains the device-independent portion of the file system. The device-dependent interface of CP/M is the BIOS (Basic I/O System) module.

coordinate scaling Transforms points from one space to another. In GSX all point coordinates must be specified in Normalized Device Coordinates with values between 0 and 32,767. GDOS then scales these coordinates into values appropriate for your graphics device.

default device driver Largest driver loaded during a graphics session. It is always the first driver named in the Assignment Table.

device driver GIOS file that translates standard device-independent graphics operations to graphics specific command sequences for a particular device. Device drivers for graphics devices are contained in the GIOS (Graphics I/O System) portion of GSX.

DR Draw Application program that provides an advanced capability to create complex graphics.

DR Graph Application program that allows you to graph and plot data by making simple menu selections.

function code Number that indicates to the operating system the function that is being requested when a service call is made.

GDOS Graphics Device Operating System, or GDOS, is the device-independent portion of GSX. It services graphics requests and calls GIOS to send commands to graphics devices.

Generalized Drawing Primitive (GDP)	A display function used to address special device capabilities such as curve drawing.
GIN	Graphics Input mode
GIOS	Graphics Input Output System, or GIOS, is the device-dependent portion of GSX. GIOS files are the individual device drivers which translate between a particular device and the standard VDI conventions.
GKS	Graphical Kernel System
graphics mode	Entered by executing the GSX command from the operating system's user interface module. This enables all graphics functions.
GSX	Graphics System Extension, or GSX, is the graphics extension to the 8080 and 8086 family of microcomputer operating systems.
Graphical Kernel System (GKS)	An international standard for the programming interface to graphics from an application program.
graphics primitives	Basic graphics operations performed by GSX; for example, drawing lines, markers, and text strings.
NDC	Normalized Device Coordinates
normalized device coordinate space	Uniform virtual space by which a graphics application program passes graphics information to a device. GDOS translates between NDC space and the Display Coordinates (DC) of a particular device.
normalized device coordinates	Virtual space in which all point coordinates are mapped to values between 0 and 32,767. NDC space serves as a common interface between graphics devices.
operation codes	Passed to GDOS as part of a parameter list; indicates which graphics operation is requested.

VDI Virtual Device Interface

virtual device interface Standard interface between device-dependent and device-independent code in a graphics environment. VDI makes all device drivers appear identical to the calling program. GSX is based on VDI, and all device drivers written for GSX must conform to the VDI specification.

workstation Graphics device with one display surface and zero or more input devices.

workstation identification number (ID) Logical unit number that specifies which graphics device is currently active. Each device driver has an associated workstation ID which is specified in an Assignment Table in file ASSIGN.SYS.

End of Glossary

Index

NOTES

Reader Comment Card

We welcome your comments and suggestions. They help us provide you with better product documentation.

Date _____

1. What sections of this manual are especially helpful?

2. What suggestions do you have for improving this manual? What information is missing or incomplete? Where are examples needed?

3. Did you find errors in this manual? (Specify section and page number.)

GSX™ Graphics Extension Programmer's Guide
Second Edition: September 1983
5000-2024

COMMENTS AND SUGGESTIONS BECOME THE PROPERTY OF DIGITAL RESEARCH.

From: _____

_____

_____

_____

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS / PERMIT NO. 182 / PACIFIC GROVE, CA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL RESEARCH™

P.O. Box 579
Pacific Grove, California
93950

Attn: Publications Production