
# GSX-80™

## Graphics Extension

# Programmer's Guide

# GSX-80™
# Graphics Extension
# Programmer's Guide

Digital Research
P.O. Box 579
160 Central
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

First Edition: November 1982
Release Number: 1.0

DIGITAL RESEARCH™

DATE: DECEMBER 1, 1982

TO: CP/M™ CARD OWNER

SUBJECT: GSX-80™ FUNCTION

Along with this CP/M Card package, you are receiving GSX-80 which is Digital Research's first graphics software product. GSX-80 is the graphic system extension to 8-bit CP/M systems and provides graphic output functions through standard O/S calling procedures. We have included it with the CP/M Card to give you the gateway to "CP/M GRAPHICS™" which includes the following products as well as more to come from independent software vendors and DRI:

**GSS-KERNEL™** is a subroutine library of 2D graphic primitives for programmers and system builders. This product will provide a programmers interface to graphics that is consistent with the emerging ISO graphic standard GKS (Graphical Kernel System).

**GSS-PLOT™** is a subroutine library of high level functions for programmers who want bar graphs, pie charts, histograms, line graphs and scatter plots. This product makes it easy to write programs that produce typical business, engineering and scientific data representation plots.

**GSS-KERNEL** and **GSS-PLOT** will link with PASCAL/MT+, PL/I-80, CB80 and FORTRAN source coded programs.

**GSS-4010™** is a interactive utility that provides Tektronix 4010 terminal emulation for microcomputers that have graphic displays. This product is for the user who wants to access graphic software packages on time sharing systems. Any software package that produces Tektronix PLOT-10 compatible output can be accessed.

These products along with GSX-80 will bring portability to microcomputer graphic applications. Many output devices are also supported such as plotters, matrix printers and CRT terminals.

See your computer retailer for these graphic products and new ones that will be coming soon as part of "CP/M GRAPHICS."

CP/M CARD™, GSX-80™, PASCAL/MT+™, PL/I-80™, CB80™, CP/M GRAPHICS™ Digital Research, Inc.

GSS-KERNEL™, GSS-PLOT™, GSS-4010™ Graphic Software Systems, Inc.

Post Office Box 579 ■ 160 Central Avenue ■ Pacific Grove, California 93950 ■ (408) 649-3896 ■ TWX 910 360 5001

# Preface

**MANUAL OBJECTIVE**

The purpose of this document is to describe the features and operation of the CP/M-80 Graphics System Extension, GSX-80. The manual will explain what GSX-80 does and how you can employ its graphics capabilities. It will also explain how GSX-80 interfaces to your hardware environment and how you can adapt GSX-80 for your own unique graphics devices.

**INTENDED AUDIENCE**

This manual is intended for systems programmers who are familiar with the CP/M Operating System and also have some knowledge of graphics programming.

**MANUAL DESIGN**

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

Section 1 provides an overview of GSX-80. It explains the GSX-80 architecture and gives a preview of each component of GSX-80. Also, it describes how to use GSX-80 in conjunction with applications programs to provide graphics capability on your system.

Section 2 describes the Graphics Device Operating System (GDOS) in detail. It includes the functions and calling conventions for GDOS as well as information about how device drivers are loaded during program execution.

Section 3 treats the Graphics Input/Output (GIOS). It describes how to interface particular graphics devices to GSX-80 to provide device independence for your application program.

Section 4 provides details about the GSX Loader and its operation at the start of your program execution. It also describes how the GSX Loader is integrated with your application program using the GENGRAF utility.

Section 5 describes the installation procedure for GSX-80 and also tells you how to debug application programs that use graphics.

Appendixes containing the following information are provided for your convenience:

Appendix A - Example graphics device driver listings
Appendix B - The Virtual Device Interface specification
Appendix C - A glossary of GSX-80 unique terms
Appendix D - A summary of device characteristics for graphics device drivers included with the standard GSX-80 distribution

Finally, an index will help you use this document more effectively.

**CONVENTIONS USED IN THIS MANUAL**

Words appearing in bold type in the main text can be found in the Glossary, Appendix C.

# Table of Contents

**SECTION 3 GIOS**



**SECTION 4 THE GENGRAF UTILITY**



**SECTION 5 OPERATING PROCEDURES**

**ILLUSTRATIONS**



**TABLES**

# Section 1
# OVERVIEW

**INTRODUCTION**

This section gives you an overview of the Graphics System Extension architecture, its components and their functions. Later sections describe each of these parts in detail.

**GRAPHICS SYSTEM EXTENSION ARCHITECTURE**

GSX-80 is the Graphics System Extension for the CP/M family of operating systems. It incorporates graphics capability into the operating system and provides a host and device independent interface for your applications programs. Graphics primitives are provided for implementing graphics applications with reduced programming effort. In addition, GSX-80 offers program portability by allowing an application to run on any CP/M system with the GSX-80 option. GSX-80 also promotes programmer portability by providing a common programmer interface to graphics which is compatible with one of the world's most widely used operating systems, CP/M.

GSX-80 is implemented as an integral part of your operating system. Application programs interface to GSX-80 through a standard calling sequence similar to the **BDOS** conventions. Drivers for specific graphics devices translate the standard GSX-80 calls to the unique characteristics of the device. In this way, GSX-80 provides device-independence since the peculiarities of the graphics device are not visible to the application program.

GSX-80 consists of several parts that work together to give your system graphics capability:

- the **Graphics Device Operating System (GDOS)**,
- the **Graphics Input/Output System (GIOS)** and,
- the **GENGRAF Utility**

## THE GRAPHICS DEVICE OPERATING SYSTEM

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 which is constant regardless of specific devices or host hardware, just as the BDOS standardizes disk interfaces. Your application program accesses GDOS through a mechanism analogous to the normal BDOS system calls.

GDOS loads at run time with your graphics application program, so it consumes system memory space only when required, leaving the normal **Transient Program Area** for non-graphic programs.

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-80 within a single application. By referring to devices with a **workstation identification number**, graphics information can be sent to any one of several 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 at any time. For details see "Loading Device Drivers" in Section 2.

## THE GRAPHICS INPUT/OUTPUT SYSTEM

The Graphics Input/Output System (GIOS) is similar to the Basic I/O system or BIOS. It provides 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 host ports. A unique device driver is required for each different graphics device on your system. The term GIOS refers to the collection of available device drivers as well as the particular driver that is loaded into memory when required by your application. Although a single program can use several graphics devices, only one driver is loaded by GDOS at a time.

GIOS performs the **graphics primitives** of GSX-80, consistent with the inherent capabilities of your graphics device. In some cases a device driver will emulate standard GDOS capabilities which are not provided by the graphics device hardware. For example, some devices may require that dashed lines be simulated by a series of short vectors generated in the device driver.

GSX-80 is supplied with drivers for many of the most popular graphics devices for microcomputer systems. However, you may install your own custom device driver if necessary. We provide information in Section 3, "GIOS," to help you write your driver, including the Virtual Device Interface (VDI) Specification which defines all the required functions and parameter conventions. In addition, we include a sample device driver in Appendix B.

THE GENGRAF UTILITY

The GENGRAF utility is used to combine your application program and the GSX Loader into one executable **.COM file.** The **GSX Loader** is a small program that loads the GDOS and GIOS into memory at run-time and establishes the links between your application program and GDOS. The GSX Loader is attached to your application program after it has been compiled/assembled and linked with the required external routines and libraries.

**THE GRAPHICS KERNEL AND PLOT**

GSX-80 defines a standard interface to graphics from your application program using a BDOS-like access method. GSX-80 also supports higher level graphics interfaces. GSS-KERNEL is a graphics application library which is based on the **Graphical Kernel System (GKS)**, level 0A, a draft international graphics standard which gives you extremely powerful graphics capabilities. GSS-KERNEL supports the popular high level languages such as Pascal, Fortran and PL/I with standard graphics procedure calls. GSS-PLOT is a high level programming tool that allows you to produce graphs and plots with just a few calls from a high level language.

**GSS-KERNEL** and **GSS-PLOT** are available for CP/M from Digital Research as separate program products. GSX-80 is the basis for these products since it provides host and device independent access to graphics primitives.

**APPLICATION PROGRAMS**

You can write your applications programs in assembly language (or a high level language that supports the GSX-80 calling conventions) with appropriate calls to GDOS. Programs are compiled/assembled and linked in the normal manner with the addition of one extra step. Before executing your application program you must attach the GSX Loader using the GENGRAF utility supplied with GSX-80. This results in an executable .COM file. See Section 4, "The GENGRAF Utility," for details.

End of Section 1

# Section 2
# GDOS

**INTRODUCTION**

In this section we describe 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 several functions during the execution of a graphics application program, including:

- trapping all system function calls,
- loading device drivers as required,
- converting normalized coordinates to device coordinates.

TRAPPING GRAPHICS CALLS

GDOS interfaces to the normal BDOS calling sequence by trapping all calls to BDOS and examining the **function code** in register C. If the code is 115, the call is a graphics request and is serviced by GDOS. Otherwise, the request is passed on to BDOS to be serviced in the normal manner. See the "GSX Loader" in Section 4 for details.

DYNAMIC LOADING

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

TRANSFORMING POINTS

All graphics coordinates are passed to GDOS as **Normalized Device Coordinates (NDC)** in a range from 0 to 32767 in both axes. Using information passed from the device driver when the workstation (device) was opened, GDOS scales the NDC coordinates to the device coordinates. The full scale NDC space is always mapped to the full dimensions of your graphics device in each axis. In this way you are assured that all your graphics information will appear on the display surface regardless of the dimensions of your device.

**GDOS CALLING SEQUENCE**

GSX-80 provides the programmer with a standard way to access graphics capabilities. This accessing method is referred to as the GSX-80 **Virtual Device Interface (VDI)** since it makes all graphics devices appear "virtually" identical. The implementation of the VDI employs the conventional BDOS calling sequence with a function code of 115. That is, the decimal number 115 (73 Hexidecimal) is loaded into register C, and a subroutine call is made to location 5. Arguments to GDOS are passed in a parameter list pointed to by the contents of the double register DE.

The parameter list is in the form 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.

The GDOS calling sequence is summarized below:

GDOS CALLING SEQUENCE:
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.)

Control Array on Output:

```
contrl(3)    -length of output coordinate array
contrl(5)    -number of vertices in output
              point array
contrl(6-n)  -opcode dependent
```

Output Parameter Array:

```
intout       -array of output parameters
```

Output Point Coordinate Array:

```
ptsout       -array of output coordinates
              (each point is specified by an X
              and Y coordinate given in Norma-
              lized Device Coordinates between
              0 and 32,767.)
```

NOTE: All array elements are type INTEGER (2 bytes). The meaning of the input and output parameter arrays is dependent on the opcode. See the "Virtual Device Interface Specification," Appendix B, for details.

**GDOS OPCODES**

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

**Table 1. GSX-80 OPERATION CODES**

| Opcode | Description |
|---|---|
| 1 | OPEN WORKSTATION- initialize a graphics device (load driver if necessary) |
| 2 | CLOSE WORKSTATION- stop graphics output to this workstation |
| 3 | CLEAR WORKSTATION- clear display device |
| 4 | UPDATE WORKSTATION- display all pending graphics on workstation |
| 5 | ESCAPE- enable special device dependent operation |
| | Escape Functions: (function id indicated in parameter list) |
| | ID 1 INQUIRE ADDRESSABLE CHARACTER CELLS- return number of addressable rows and columns |
| | 2 ENTER GRAPHICS MODE- enter graphics mode |
| | 3 EXIT GRAPHICS MODE- exit graphics mode |
| | 4 CURSOR UP- move cursor up one row |
| | 5 CURSOR DOWN- move cursor down one row |
| | 6 CURSOR RIGHT- move cursor right one column |





1C

**Table 1. GSX-80 OPERATION CODES (continued)**

| Opcode | Description |
|---|---|
| | 7 CURSOR LEFT- move cursor left one column |
| | 8 HOME CURSOR- move cursor to home position |
| | 9 ERASE TO END OF SCREEN- erase from current cursor position to end of screen |
| | 10 ERASE TO END OF LINE- erase from current cursor position to end of line |
| | 11 DIRECT CURSOR ADDRESS- move alpha cursor to specified row and column |
| | 12 OUTPUT CURSOR ADDRESSABLE TEXT- output text at the current alpha cursor position |
| | 13 REVERSE VIDEO ON- display subsequent text in reverse video |
| | 14 REVERSE VIDEO OFF- display subsequent text in standard video |
| | 15 INQUIRE CURRENT CURSOR ADDRESS- return location of alpha cursor |
| | 16 INQUIRE TABLET STATUS- return status of graphics tablet |
| | 17 HARDCOPY- make hardcopy |
| | 18 PLACE CURSOR AT LOCATION- move cursor directly to specified location |
| | 19 REMOVE CURSOR- do not display cursor |

**Table 1. GSX-80 OPERATION CODES (continued)**

| Opcode | Description |
|---|---|
| | 20-<br>50 RESERVED (for future expansion) |
| | 51-<br>100 UNUSED (and available) |
| 6 | POLYLINE- output a polyline |
| 7 | POLYMARKER- output markers |
| 8 | TEXT- output text starting at specified position |
| 9 | FILLED AREA- display and fill a polygon |
| 10 | CELL ARRAY- define a cell array |
| 11 | GENERALIZED DRAWING PRIMITIVE- display a generalized drawing primitive |
| | ID 1 BAR |
| | 2 ARC |
| | 3 PIE SLICE |
| | 4 CIRCLE |
| | 5 PRINT GRAPHIC CHARACTERS |
| | 6-7 RESERVED (for future use) |
| | 8-10 UNUSED (and available) |
| 12 | SET CHARACTER HEIGHT- set text size |





10

**Table 1. GSX-80 OPERATION CODES (continued)**

| Opcode | Description |
|---|---|
| 13 | SET CHARACTER UP VECTOR- set text direction |
| 14 | SET COLOR REPRESENTATION- define the color associated with a color index |
| 15 | SET POLYLINE LINETYPE- set linestyle for polylines |
| 16 | SET POLYLINE LINE WIDTH- set width of lines |
| 17 | SET POLYLINE COLOR INDEX- set color for polylines |
| 18 | SET POLYMARKER TYPE- set marker type for polymarkers |
| 19 | SET POLYMARKER SCALE- set size for polymarkers |
| 20 | SET POLYMARKER COLOR INDEX- set color for polymarkers |
| 21 | SET TEXT FONT- set device dependent text style |
| 22 | SET TEXT COLOR INDEX- set color of text |
| 23 | SET FILL INTERIOR STYLE- set interior style for polygon fill |
| 24 | SET FILL STYLE INDEX- set fill style for polygons |
| 25 | SET FILL COLOR INDEX- set color for polygon fill |
| 26 | INQUIRE COLOR REPRESENTATION- return color representation values of index |

**Table 1. GSX-80 OPERATION CODES (continued)**

| Opcode | Description |
|---|---|
| 27 | INQUIRE CELL ARRAY- return definition of cell array |
| 28 | INPUT LOCATOR- return value of locator |
| 29 | INPUT VALUATOR- return value of valuator |
| 30 | INPUT CHOICE- return value of choice device |
| 31 | INPUT STRING- return character string |
| 32 | SET WRITING MODE- set current writing mode (replace, overstrike, complement, erase) |
| 33 | SET INPUT MODE- set input mode (request or sample) |

## LOADING DEVICE DRIVERS

The GSX-80 Virtual Interface refers to graphics devices as workstations. Each time a graphics device is to be used, it must first be initialized with an OPEN WORKSTATION operation. This will cause the device to be initialized with selected attributes such as line type, color, etc., and it will also return information about the device to GDOS.

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

If there is not a match, then GDOS must load the correct device driver. In order to do this, GDOS refers to a data structure called the **Assignment Table** which contains information about the available device drivers and the files where they are stored.

GDOS searches the Assignment Table for a device driver entry with a driver number which matches the workstation ID requested in the OPEN WORKSTATION call. If it finds the correct driver entry, GDOS will load the new device driver above itself where the previous driver was located. When the load is complete, GDOS will finish the OPEN WORKSTATION operation and then return to the calling program.

If there is no match in the Assignment Table when a new driver is required, GDOS will return without loading a driver. Therefore the previous graphics device will continue to be the open workstation.

ASSIGNMENT TABLE FORMAT

The Assignment Table consists entirely of text and may be created or modified with any text editor. It must reside in a file named ASSIGN.SYS on the currently logged drive when GSX-80 is operating. For each device driver, there is an entry containing the driver number, which signifies 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 may be any legal CP/M unambiguous filename. Any device to be 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:

DDXd:filename

DD = Logical driver number
X = space
d = Disk drive code
filename = the driver filename (valid unambiguous CP/M filename of up to eight upper case characters, .PRL extension required)

For example, a valid entries in the Table would be:
```
11 A:DDPLOT
 1 B:CRTDRV
21 A:PRINTR
```

---

There is a convention for assigning device driver numbers (workstation ID's) to graphics devices. This is done to assure the maximum degree of device independence within applications programs. The convention for driver numbers is:

| | |
|---|---|
| 0 | Default console |
| 1-10 | CRT |
| 11-20 | Plotter |
| 21-30 | Printer |
| 32-40 | Other devices |

---

MEMORY MANAGEMENT

When execution of your graphics program begins, the GSX Loader allocates memory for the first device driver in the Assignment Table at the top of the Transient Program Area, just below BDOS. 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. To prevent new drivers from writing over and destroying a portion of BDOS, which follows the device driver, make sure 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. An error is reported if an attempt is made to load a driver larger than the default driver.

End of Section 2

# Section 3
# GIOS

**INTRODUCTION**

In this section we describe the Graphics Input/Output System, or GIOS. The information in this section will allow you to write and install your own custom drivers for unique graphics devices.

NOTE: If your disk does not include all the GIOS modules documented in this manual, contact your OEM or distributor.

**THE PURPOSE OF GIOS**

As we discussed earlier, GSX-80 is composed of three components: the Graphics Device Operating System (GDOS), the Graphics Input/Output System (GIOS), and the GSX Loader. GDOS contains the device independent graphics functions, while GIOS contains the device dependent code. This division is consistent with the CP/M philosophy of isolating device dependencies so that the principal parts of the operating system are transportable to many systems and so that applications can run independent of the specific devices connected to the system. In this context, GIOS is analogous to the BIOS but pertains to graphics devices only. GIOS contains a device driver for each of the graphics devices on the system.

A difference between GIOS and BIOS is that whereas all device drivers contained within BIOS 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 may be changed by a request from the application program. GDOS insures

that the correct driver is in memory when required. Because GIOS drivers are loaded dynamically, they must be stored on disk in relocatable format, as .PRL file types.

**DEVICE DRIVER FUNCTIONS**

Device Drivers use the intrinsic graphics capabilities of devices to implement graphics primitives for GDOS. In some cases the graphics device itself does not support all the GDOS operations directly and the driver must emulate the capability in software. As an 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-80 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.

Arguments to device drivers are passed in a parameter list pointed to by the contents of the double register DE. The parameter list is in the form 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 a device driver is indicated by an operation code in the parameter list.

The device driver calling sequence is summarized below:

DEVICE DRIVER CALLING SEQUENCE:

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 Device Coordinates) |

Control Array on Output:

```
contrl(3)    -length of output coordinate array
contrl(5)    -number of vertices in output
               point array
contrl(6-n)  -opcode dependent
```

Output Parameter Array:

```
intout       -array of output parameters
```

Output Point Coordinate Array:

```
ptsout       -array of output coordinates
               (each point is specified by an X
               and Y coordinate given in Device
               Coordinates)
```

All array elements are type INTEGER (2 bytes). The meaning of the input and output parameter arrays is dependent on the opcode. See the Virtual Device Interface Specification, Appendix B, for details.

All graphics coordinates are passed to the device driver as Device Coordinates. Using information passed from the device driver when the workstation (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. In this way you are assured that all of your graphics information is visible on the display surface regardless of the actual device dimensions.

If your device has an aspect ratio that is not 1:1 (i.e., the display surface is not square), and you wish to prevent distortion between your world coordinate system and the device coordinate system, then in your application you 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 non-critical applications you need not make this adjustment.

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

**CREATING A GIOS**

Device driver files that are part of GIOS must be in relocatable format so they can be loaded by the GSX Loader and GDOS. You may 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 using LINK-80 to produce a relocatable load module.

When naming the device driver file the name used must contain six characters and must have a .PRL extension. In addition, if the driver is to be used by a graphics application, it must be included in the Assignment Table. This is a text file named ASSIGN.SYS on the currently logged disk. Each device entry in the Assignment Table has the form:

DDXd:filename

where DD = device driver number (workstation ID)
X = a space
d = the disk drive code
filename = the driver filename (valid unambiguous CP/M filename--up to eight upper case characters required)

See "Assignment Table Format" in Section 2 on GDOS for more details.

End of Section 3

# Section 4
# THE GENGRAF UTILITY

**INTRODUCTION**

In this section we describe the GSX-80 utility GENGRAF and the GSX Loader. The GSX Loader brings GDOS into memory in preparation for execution of your graphics application program. GENGRAF is used to attach the GSX Loader to your program creating an executable .COM file.

**THE GSX LOADER**

GDOS is not resident in memory when the operating system is initialized, but is loaded when you execute your graphics application program. In this way, maximum space is preserved for non-graphics programs.

Loading of GDOS at run-time is performed by a special program called the GSX Loader. The GSX Loader also brings the Assignment Table into memory from a file named ASSIGN.SYS on the currently logged disk and then allocates memory space for the first (default) device driver named in the Assignment Table. Finally, the GSX Loader establishes the linkage between GDOS and the normal BDOS entry point at location 5 by moving some pointers.

The GSX Loader comes into memory with your graphics application program and receives control before execution of your program begins. The procedure for attaching the GSX Loader to your program is explained below.

GSX LOADER RUN-TIME PROCEDURES

After the GSX Loader is brought into memory with your program (through the normal operation of the Console Command Processor) it immediately receives control and performs the following operations.

First, the GSX Loader opens the Assignment Table file, ASSIGN.SYS, and reads it to determine the filename of the default device driver. It then allocates space for the default device driver into the highest portion of the Transient (user) Program Area, just below BDOS. This defines the space allocated for all device drivers, since new drivers loaded during execution of your program will be overlayed onto the current driver; therefore, the default device driver must be the largest driver needed during execution of a specific application. The GSX Loader utilizes the system .PRL loader to do this (all device driver files must be in .PRL format).

Secondly, the GSX Loader brings GDOS into memory from file GSX.SYS on the currently logged disk and places it below the device driver.

Then the GSX Loader places the Assignment Table into a data area within GDOS. It also establishes the linkage from GDOS to the rest of the operating system in such a way that causes all operating system calls to go to GDOS. GDOS will filter out the graphics calls (Function 115) and execute them and allow the BDOS calls (all other Function codes) to pass through to BDOS unaltered.

Note that the new top of the Transient Program Area, or TPA, (indicated by the vector in location 5) is the bottom of GDOS, so that the application program will be aware of the actual free space available and allocate stack areas, etc., below GDOS.

Finally, the GSX Loader moves the application program down from its position above the GSX Loader to the start of TPA, location 100H. The space that was taken up by the Loader can be utilized during the execution of the application program. Control is then transferred to location 100H, the start of the application program. See Figure 1.

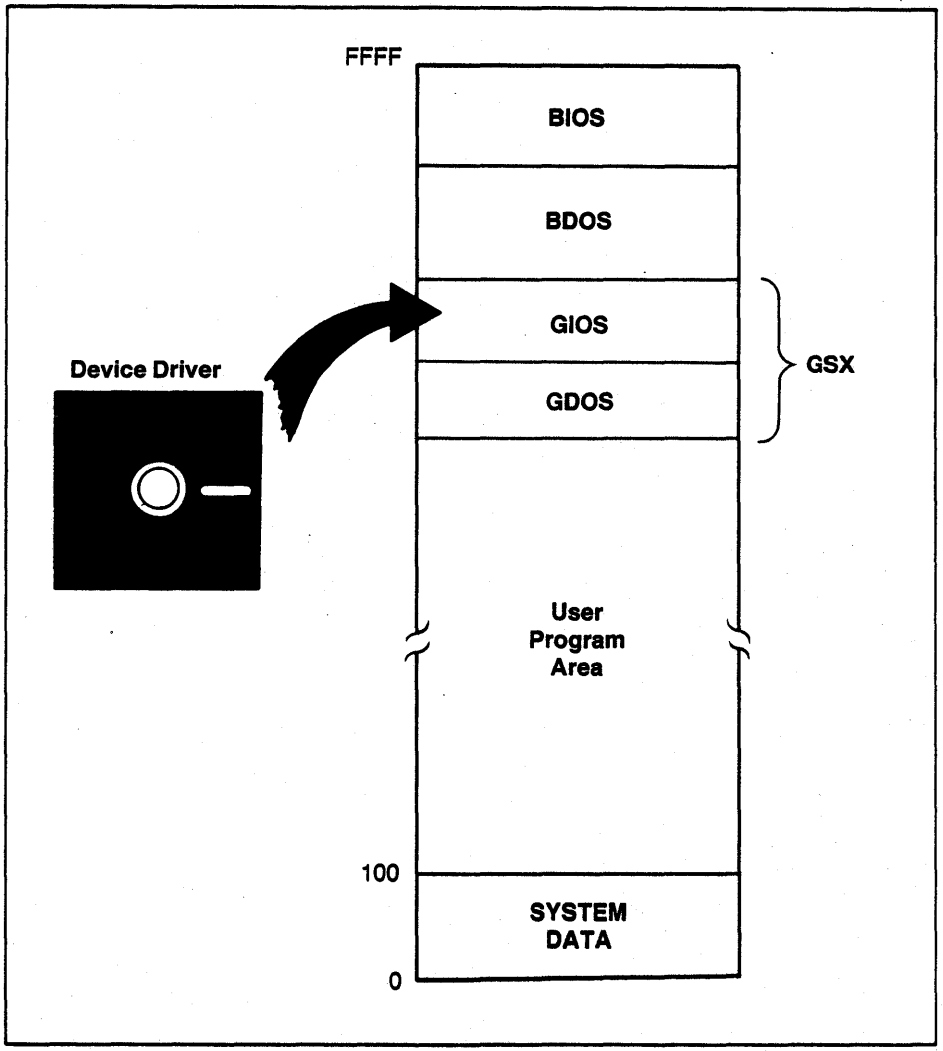


**Figure 1. GSX-80 Memory Map.**

## THE GENGRAF UTILITY

Your application program may be written in any language provided the GDOS protocol is observed. You may compile/assemble and link your application in the normal manner, yielding a .COM executable file. One additional step must be performed, however, before executing your graphics program: the GSX Loader must be attached to the front of your program so that it can prepare the operating system environment for your graphics application.

The GENGRAF utility (provided with the GSX-80 distribution) allows you to attach the loader to your program with one simple command:

GENGRAF <filename>

For example, if your graphics application program were in an executable file named MYFILE.COM, then the following command string would attach the GSX Loader and place the result into file MYFILE.COM.

GENGRAF MYFILE

The resulting MYFILE.COM file would be ready to run.

You should be aware of the total memory space available to your application program in the TPA. This will be less for graphics applications than for normal programs because of the GDOS and device driver requirements. We will explain how to calculate the exact size of the TPA in Section 5, "Operating Procedures."

End of Section 4

# Section 5
# OPERATING PROCEDURES

**INTRODUCTION**

This section explains how to employ GSX-80 in your graphics applications. It also gives you some aids for debugging your application programs and determining the user program area available on your system. Finally, it describes how to create and install a new graphics device driver under GSX-80.

**GSX-80 DISTRIBUTION FILES**

When you receive your GSX-80 distribution diskette, first check to insure that all required files have been included. The following files should be present on your diskette:

ASSIGN.SYS this is the default assignment table file which associates specific device drivers with logical device numbers.

GSX.SYS this is the GSX-80 executable file.

GENGRAF.COM this is the GENGRAF utility file which is used to attach the GSX Loader to your application program.

DDxxxx.PRL there will be one file with this naming convention for each device driver. The xxxx will be unique for each device and will usually be derived from the device tradename or model number.

If any files are missing, contact your distributor to receive a new diskette. If all files are present, make a backup of the distribution diskette using the PIP utility and store your distribution diskette in a safe place. Then, using the backup diskette, transfer the GSX-80 files to your working system diskette. Always use the backup diskette to generate any new copies of GSX-80. Do not use the distribution diskette for routine operations.

## RUNNING GRAPHICS APPLICATIONS UNDER GSX-80

In order to use the graphics features Provided by GSX-80, you must insure that several conditions are met:

1. In your application program you must conform to the GSX-80 calling convention to access graphics primitives. This involves making a call to the operating system in the normal manner with a function code of 115 (115 in register C, CALL to location 5). In addition, an address must be placed in register DE when the call is made which points to a parameter list that provides information to GSX-80 and also returns information to the calling program. The details of this procedure are contained in the sections on GDOS and GIOS and in Appendix B, "The Virtual Device Interface Specification."

2. After successfully compiling/assembling and linking your application program, you must perform one additional step before running your program. Using the GENGRAF utility provided with GSX-80, you must append the GSX Loader to your program. This is done with a simple command string. If your program was named MYFILE.COM on the currently logged disk, then the command would appear as follows:

```
A>GENGRAF MYFILE
```

The .COM extension is assumed by GENGRAF.

3. You must insure that the required device drivers are present on the currently logged disk when your program is executed. Also, the Assignment Table must contain the names of your device drivers and a logical device number or workstation ID which corresponds to the correct device driver. The details of device driver and Assignment Table requirements are included in Section 2, "GDOS," and Section 3, "GIOS."

**DEBUGGING GRAPHICS APPLICATIONS UNDER GSX-80**

The GSX Loader, attached to your application using GENGRAF, loads GSX-80 from the disk along with the default device driver. It then moves the actual application code down to the normal TPA start address at location 100H (refer to Section 4, "The GENGRAF Utility"). To debug a program using SID or DDT, type the following:

```
SID MYPROG.COM MYPROG.SYM
                or
DDT MYPROG.COM
```

The debugger will respond with a prompt: *

First, you must determine where GSX-80 is located. This will also tell where the top of the TPA area is. Type the following to start the GSX Loader and break before the application program is moved down to location 100H:

```
*G,103
```

To find out where GSX-80 is located you must look at the GDOS jump vector at locations 5 through 7 (JMP <address>). Type:

```
*D5,7
```

The debugger will display the contents of locations 5 through 7:

C3 00 9B

This indicates that GSX-80 is at location 9B00 (C3 is the JMP opcode). The last instruction location before the end of the TPA is the GSX-80 location minus 3, in this case 9B00-3 = 9AFD.

NOTE: Your TPA may be in a different place. Always subtract 3 from the contents of address 6,7 to determine the top of the TPA.

Now, to view your program before it begins execution, set a breakpoint that will occur when the last location of the TPA (determined above) is moved down by the GSX Loader:

*G,9AFD

Then you may list your program which is now located at 100H:

*L100

**DETERMINING USER PROGRAM AREA SIZE**

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

Size of TPA in bytes = (Size of GSX.SYS) +

[8 * (Size of PRL file of largest device driver used during the application)] / 9 +

(Size of application COM file after running GENGRAF) + 20

## CREATING A NEW DEVICE DRIVER

GSX-80 is distributed with a number of device drivers for popular graphics devices. If your devices are included (refer to Appendix D, "Device Specifics," for a summary of the supported devices) then you only need to edit the Assignment Table file with a text editor to make sure that it reflects the logical device numbering assignments that you desire. However, if your device is not supported, you must create a driver program for your device which conforms to the VDI specification. This may be written 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 convention. 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 may be generated by the driver if they are not directly available in the device. The complete VDI specification is given in Appendix B and the parameter passing convention is discussed in Section 2, "GDOS" and Section 3, "GIOS."

To help you design and code your own device driver, we have included a driver skeleton in Appendix A which you can use as a boilerplate. In addition we have listed several device drivers as an example for you to follow.

After coding assembling and linking your device driver, you will have a .PRL file. To make this driver known to GSX-80, 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.

End of Section 5

# Appendix A
# EXAMPLE DEVICE DRIVER

We have included an example to help guide you through the design of a GSX-80 device driver. The following is a listing of the device driver for the Retrographics Video Terminal. It conforms to the GSX-80 Virtual Device Interface and is written in RATFOR (Structured Fortran). Refer to Section 3,"GIOS," and Appendix B, "Virtual Device Interface Specification," for more information on device drivers.

```
subroutine ddvret (contrl, intin, ptsin, intout, ptsout)
#########################################################################
#         THIS MATERIAL IS CONFIDENTIAL AND IS FURNISHED UNDER          #
#         A WRITTEN LICENSE AGREEMENT.  IT MAY NOT BE USED,             #
#         COPIED OR DISCLOSED TO OTHERS EXCEPT IN ACCORDANCE            #
#         WITH THE TERMS OF THAT AGREEMENT.                             #
#                                                                       #
#         COPYRIGHT (C) 1982 GRAPHIC SOFTWARE SYSTEMS INC.              #
#         ALL RIGHTS RESERVED.                                          #
#                                                                       #
#     Function: Device driver for VT100 with Retrographics              #
#     Input Parameters:                                                 #
#           contrl      - An integer array with following information #
#                         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             #
#     Routines Called:                                                #
#           dcvret - change color on Retrographics terminal           #
#           xy40xx - output x,y coordinate on 40xx terminal           #
#           mult   - multiply 2 16-bit numbers                        #
#           dm40xx - 40xx marker emulation routine                    #
#           gdevot - put a character to device                        #
#           gdstin - get a string from the current device             #
#           gdstot - output a string to the current device            #
#           gimnmx - bound an integer variable                        #
#           gitoch - convert integer to character string              #
#           gchtoi - convert character string to integer              #
#########################################################################
```

```
define(`XxLIMITSx4010',1023)
define(`YxLIMITSx4010',779)
define(`DEFAULT',1)                    # Default input device
define(`CROSSHAIRS',2)                 # Crosshairs input device
define(`REPLACExMODE',1)               # Replace writing mode
define(`XORxMODE',3)                   # Xor writing mode
define(`ERASExMODE',4)                 # Erase writing mode
define(`STRINGxINPUT',4)               # String input class

integer contrl(1),intin(1),ptsin(1),intout(1),ptsout(1)

SHORTINT opcode
integer alfamd(2), clrwrk(5), i, j, index, gimnmx, xy(4), tries,
       ginok, chrhgt(4), hgtin, line(8), xhi, xlo, yhi, ylo, booboo(2),
       lodcur(4), enable(4), setup(5), itemp, chrwid(4), xcoord, ycoord,
       celwid(4), celhgt(4), curup(3), curdwn(3), currgt(3), curlft(3),
       curhom(3), erscrn(3), erslin(3), revon(4), revoff(4), entgrf(3),
       extgrf(2), inqcur(5), k, gitoch

integer mult

integer ccltb(2), sclrd(2), sclgr(2), sclbl(2), clrflg

include(`ddcom')

common /cmvret/ ccltb, sclrd, sclgr, sclbl, clrflg

# The following equivalence statements are used to decrease the amount of code
#    necessary to access specific array elements. The arrays and the
#    variables equivalenced are listed below:
#
#       line(2) :: xhi
#       line(3) :: xlo
#       line(4) :: yhi
#       line(5) :: ylo
equivalence (line(2), xhi), (line(3), xlo), (line(4), yhi), (line(5), ylo)
equivalence (xcoord, xy(1)) , (ycoord, xy(2))

data celwid /13, 26, 39, 51/    # Char cell width in raster space
data celhgt /23, 46, 71, 98/    # Char cell height in raster space
data chrwid / 9, 17, 26, 34/    # Actual char width 2/3 * celwid
data chrhgt /14, 28, 43, 59/    # Actual char height .6 * celhgt

#    move cursor up 1 row
data curup /ESC, LBRACK, BIGA/
```

```
#    move cursor down 1 row
data curdwn /ESC, LBRACK, BIGB/

#    move cursor right 1 row
data currgt /ESC, LBRACK, BIGC/

#    move cursor left 1 row
data curlft /ESC, LBRACK, BIGD/

#    move cursor home (upper left hand corner of screen)
data curhom / ESC, LBRACK, BIGH/

#    erase to end of screen
data erscrn /ESC, LBRACK, BIGJ/

#    erase to end of line
data erslin /ESC, LBRACK, BIGK/

#    reverse video on
data revon /ESC, LBRACK, DIG7, LETM/

#    reverse video off
data revoff /ESC, LBRACK, DIG0, LETM/

#   enter graphics mode from alpha cursor mode
data entgrf /GS, US, NEWLINE/

#   exit graphics mode into alpha cursor mode
data extgrf /CAN, NEWLINE/

#    inquire current cursor address
data inqcur /ESC, LBRACK, DIG6, LETN, NEWLINE/

#    put back in alpha mode
data alfamd /US,NEWLINE/

#    booboo
data booboo /BELL, NEWLINE/

#    load cursor
data lodcur /ESC, SLASH, LETF, NEWLINE/

#    clear workstation and enquire status to keep from overflowing
data clrwrk /ESC, FF, ESC, ENQ, NEWLINE/

#    enable GIN
data enable /BELL, ESC, SUB, NEWLINE/
```

```
#    setup - initialize device
data setup /GS, US, ESC, 0, NEWLINE/


contrl(3) = 0    # Initialize ptsout count to zero (0)

opcode = contrl(OPCODE)    # Obtain a local copy of the current opcode

#
#  opcode open workstation
#
if (opcode == OPENxWORKSTATION) {

      contrl(3) = 6                          # Set to number of vertices in ptsout
      contrl(5) = 45                         # Set to number of output parameters

      ndlntp = intin(2)                          # Set current device line style
      if (ndlntp < 1 | ndlntp > 5) ndlntp = 1
      ndlntp = ndlntp + UNDERLINE                # Save actual ascii character
      ndclrl = gimnmx (intin(3), 0, 1)           # current polyline color
      ndmktp = intin(4)                          # Set current polymarker type
      if (ndmktp < 1 | ndmktp > 5) ndmktp = 3 # use default
      ndclrm = gimnmx (intin(5), 0, 1)           # current polymarker color
      ndclrt = gimnmx (intin(7), 0, 1)           # current text color
      ndclrf = gimnmx (intin(10), 0, 1)          # current fill area color
      ndclrp = -1                                # no current color
      ndlcmd = REQUESTxMODE                      # locator input mode is request
      ndvlmd = REQUESTxMODE                      # valuator input mode is request
      ndchmd = REQUESTxMODE                      # choice input mode is request
      ndstmd = REQUESTxMODE                      # string input mode is request

      intout(1) = XxLIMITSx4010   # Addressable width in rasters of screen
      intout(2) = YxLIMITSx4010   # Addressable height in rasters of screen
      intout(3) = OTHER  # Device coordinates in raster units
```





71

```
# micrometers per raster along the x axis
#   the screen area is 15.24 cm high and 20.32 cm wide therefore
#   the raster size is 203000/1024 in x and 152000/780 in y
intout(4) = 198
intout(5) = 195
intout(6) = 4       # Number of character heights
intout(7) = 5       # Number of line types
intout(8) = 1       # Number of line widths
intout(9) = 5       # Number of marker types
intout(10) = 1       # Number of marker height
intout(11) = 1      # Number of fonts
intout(12) = 0      # Number of patterns
intout(13) = 0      # Number of hatch styles
intout(14) = 2      # Number of predefined colors
intout(15) = 0      # Number of GDPs
do i = 16,25 {
   intout(i) = -1     # List of GDPs
   intout(i+10) = -1 # List of associated bundle tables
   }
intout(36) = MONOCHROME  # Color capability flag
intout(37) = NO          # Text rotation capability flag
intout(38) = NO          # Fill area capability flag
intout(39) = NO          # Pixel operation capability flag
intout(40) = 2           # Number of available colors
intout(41) = 1           # Number of locator devices
intout(42) = 0           # Number of valuator devices
intout(43) = 0           # Number of choice devices
intout(44) = 1           # Number of string devices
intout(45) = 2           # Workstation type

ptsout(1) = 0
ptsout(2) = chrhgt(1)  # Minimum character height in device coordinates
ptsout(3) = 0
ptsout(4) = chrhgt(4)  # Maximum character height in device coordinates
ptsout(5) = 1      # Minimum line width in NDC space
ptsout(6) = 0
ptsout(7) = 1      # Maximum line width in NDC space
ptsout(8) = 0
ptsout(9) = 0
ptsout(10) = 12    # Minimum marker height in NDC space
ptsout(11) = 0
ptsout(12) = 12    # Maximum marker height in NDC space
```

```
      # initialize predefined color table
      ccltb(1) = 0   # Current color table
      ccltb(2) = 1000
      # the user set color table, in case of inquiry
      sclrd(1) = 0
      sclrd(2) = 1000
      sclgr(1) = 0
      sclgr(2) = 1000
      sclbl(1) = 0
      sclbl(2) = 1000
      # initialize color representation flag
      clrflg = NO

      # put device in retrographics mode, and
      # set the line style to the current style

      call gioini (1)           # initialize i/o system for crt device
      setup(4) = ndlntp         # set line style
      call gdstot (5, setup)
   }

#
#  opcode CLOSExWORKSTATION
#
else if (opcode == CLOSExWORKSTATION) {
      call gdevot (CAN)
      call gdevot (NEWLINE)
      call giostp               # close i/o system
      }

#
#  opcode CLEARxWORKSTATION
#
else if (opcode == CLEARxWORKSTATION) {
      call gdstot (5, clrwrk)
      call gdstin (6, line, i)
      }

#
#  opcode UPDATExWORKSTATION
#
else if (opcode == UPDATExWORKSTATION) {
      call gdevot (NEWLINE)
      }
```

```
#
#  opcode ESCAPE
#
else if (opcode == ESCAPE) {
     opcode = contrl(6)  # Get the escape sub opcode
     if (opcode == INQUIRExADDRESSABLExCELLS) {
        intout(1) = 24
        intout(2) = 30
        }
     else if (opcode == ENTERxGRAPHICSxMODE)  {
        call gdstot (3, entgrf)
        }
     else if (opcode == EXITxGRAPHICSxMODE)  {
        call gdstot (2, extgrf)
        }
     else if (opcode == CURSORxUP)  {
        call gdstot (3, curup)
        }
     else if (opcode == CURSORxDOWN)  {
        call gdstot (3, curdwn)
        }
     else if (opcode == CURSORxRIGHT)  {
        call gdstot (3, currgt)
        }
     else if (opcode == CURSORxLEFT)  {
        call gdstot (3, curlft)
        }
     else if (opcode == HOMExCURSOR)  {
        call gdstot (3, curhom)
        }
     else if (opcode == ERASExTOxENDxOFxSCREEN)  {
        call gdstot (3, erscrn)
        }
     else if (opcode == ERASExTOxENDxOFxLINE)  {
        call gdstot (3, erslin)
        }
     else if (opcode == DIRECTxCURSORxADDRESS)  {
        call gdstot (2, curhom)   #  Position cursor command
        i = gimnmx (intin(1), 1, 24)  # Set the row
        j = gitoch (i, line, 2, k)
        call gdstot (j, line)
        call gdevot (SEMICOL)
        i = gimnmx (intin(2), 1, 80)  # Set the column
        j = gitoch (i, line, 2, k)
        call gdstot (j, line)
        call gdevot (BIGH)
        }
```

```
    else if (opcode == OUTPUT CURSOR ADDRESSABLE TEXT) {
       call gdstot (contrl(4), intin)
       }
    else if (opcode == REVERSE VIDEO ON)  {
       call gdstot (4, revon)
       }
    else if (opcode == REVERSE VIDEO OFF)  {
       call gdstot (4, revoff)
       }
    else if (opcode == INQUIRE CURRENT CURSOR ADDRESS)  {
       call gdstot (5, inqcur)
       call gdevin (i)                        # Skip first 2 chars ESC [
       call gdevin (i)
       i = 0
       repeat {                               # Read until terminator, R is found
          call gdevin (j)
          i = i + 1
          line(i) = j
          } until (j == BIGR)
       call gchtoi (line, 1, intout(1), j)  # Convert row number
       j = j + 1                              # Bypass terminator in string
       call gchtoi (line, j, intout(2), i)  # Convert column number
       }
    }

#
#  opcode POLYLINE
#
else if (opcode == POLYLINE) {
   call dcvret (ndclrl)      # change color to current line color

   j = 1
   call gdevot (GS)              # Move to first point
   for (i=1; i<=contrl(2); i=i+1) {
      call xy40xx (ptsin(j), ptsin(j+1))
      j = j + 2
      }
   call gdstot (2, alfamd)   # Put back in alpha mode
   }
```





2D

```
#
#  opcode polymarker
#
else if (opcode == POLYMARKER) {
   call dcvret (ndclrm)        # change color to current marker color

   call gdevot (ESC)
   call gdevot (ACCENT)        # Set solid line style

   j = 1
   for (i=1; i<=contrl(2); i=i+1) {
      xy(1) = ptsin(j)     # x coordinate of marker
      xy(2) = ptsin(j+1)   # y coordinate of marker
      call dm40xx (xy)
      call gdstot (2, alfamd)    # Put back in alpha mode
      j = j + 2
      }

   call gdevot (ESC)
   call gdevot (ndlntp)                          # Restore current line style
   }

#
#  opcode text
#
else if (opcode == TEXT) {
   call dcvret (ndclrt)                 # change color to current text color

   call gdevot (GS)                     # Do a move to point to output text
   call xy40xx (ptsin(1), ptsin(2))
   call gdstot (2, alfamd)              # Put back in alpha mode
   call gdstot (contrl(4), intin)
   }

#
#  opcode fill area
#
else if (opcode == FILLxAREA) {
   call dcvret (ndclrf)                  # change color to current fill area color
```

```
   j = 1
   call gdevot (GS)                    # Move to first point
   for (i=1; i<=contrl(2); i=i+1) {
      call xy40xx (ptsin(j), ptsin(j+1))
      j = j + 2
      }
   call gdstot (2, alfamd)             # Put back in alpha mode
   }

#
#  opcode cell array
#
else if (opcode == CELLxARRAY) {

   # This device can't do pixel arrays very easily, so outline the area
   # in the current line color

   call dcvret (ndclrl)                # Change color to line color
   call gdevot (ESC)                   # Set line type to solid
   call gdevot (ACCENT)

   xlo = ptsin(1)
   ylo = ptsin(2)
   xhi = ptsin(3)
   yhi = ptsin(4)
   call gdevot (GS)
   call xy40xx (xlo, ylo)
   call xy40xx (xhi, ylo)
   call xy40xx (xhi, yhi)
   call xy40xx (xlo, yhi)
   call xy40xx (xlo, ylo)
   call gdstot (2, alfamd)             # Put back in alpha mode

   call gdevot (ESC)                   # Restore line type
   call gdevot (ndlntp)
   }
```





23

```
#
#  opcode set character height
#
else if (opcode == SETxCHARACTERxHEIGHT) {
      hgtin = ptsin(2)                # Get requested height
      ndtxsz = 1
      repeat {
         if (chrhgt(ndtxsz) > hgtin) break
         ndtxsz = ndtxsz + 1
         } until (ndtxsz > 4)
      ndtxsz = ndtxsz -1
      ndtxsz = gimrmx (ndtxsz, 1, 4)
      call gdevot (ESC)
      call gdevot (ndtxsz+SLASH)# character size is 1='0', 2='1', 3='2', 4='3'
      contrl(3) = 2                   # Set the number vertices
      ptsout(1) = chrwid (ndtxsz)     # Return values selected
      ptsout(2) = chrhgt (ndtxsz)
      ptsout(3) = celwid (ndtxsz)
      ptsout(4) = celhgt (ndtxsz)
   }

#
#  opcode color
#
else if (opcode == SETxCOLORxREPRESENTATION) {
      i = gimrmx (intin(1), 0, 1) + 1  # Map index 0-1 to 1-2

      clrflg = YES # Inform color routine that a representation
                   # has changed

      #  If all are set to 0, then he wants the background color
      j = intin(2) + intin(3) + intin(4)
      if (j == 0)  ccltb(i) = 0
      else  ccltb(i) = 1000  # The foreground color

      sclrd(i) = intin(2)  # This is what was set
      sclgr(i) = intin(3)
      sclbl(i) = intin(4)
      }
```

```
#
#  opcode set polyline linetype
#
else if (opcode == SETxPOLYLINExLINETYPE) {
      #  i = 1  ACCENT
      #       2  A
      #       3  B
      #       4  C
      #       5  D
      ndlntp = intin(1)
      if (ndlntp > 5) ndlntp = 1
      ndlntp = gimnmx (ndlntp, 1, 5) # 4012 has 5 line styles, 1-5
      intout(1) = ndlntp             # Return linestyle seleced
      ndlntp = ndlntp + UNDERLINE
      call gdevot (ESC)
      call gdevot (ndlntp)
      }

#
#  opcode set polyline colour index
#
else if (opcode == SETxPOLYLINExCOLORxINDEX) {
      ndclrl = gimnmx (intin(1), 0 ,1)
      intout(1) = ndclrl
      }

#
#  opcode set polymarker type
#
else if (opcode == SETxPOLYMARKERxTYPE) {
      ndmktp = intin(1)
      if (ndmktp < 1 | ndmktp > 5) ndmktp = 3   # Out of range defaults to 3
      intout(1) = ndmktp
      }

#
#  opcode set polymarker colour index
#
else if (opcode == SETxPOLYMARKERxCOLORxINDEX) {
      ndclrm = gimnmx (intin(1), 0 ,1)
      intout(1) = ndclrm
      }
```

```
#
#  opcode set text color index
#
else if (opcode == SETxTEXTxCOLORxINDEX) {
      ndclrt = gimnmx (intin(1), 0 ,1)
      intout(1) = ndclrt
      }

#
#  opcode set fill area color index
#
else if (opcode == SETxFILLxCOLORxINDEX) {
      ndclrf = gimnmx (intin(1), 0 ,1)
      intout(1) = ndclrf
      }
#
#  opcode inquire colour representation
#
else if (opcode == INQUIRExCOLORxREPRESENTATION) {
   i = gimnmx (intin(1), 0, 1) + 1  # Map index 0-1 to 1-2
   intout(1) = i - 1  #  This is what we inquired on
   index = intin(2)   # Type of inquiry, 0=set, 1=realized
   if (index == 0)  {
      intout(2) = sclrd(i)
      intout(3) = sclgr(i)
      intout(4) = sclbl(i)
      }
   else {
      if (index == 1)  j = ccltb(i)  # inquire realized color
      intout(2) = j
      intout(3) = j
      intout(4) = j
      }
   }
```

```
#
#  opcode input locator
#
else if (opcode == INPUTxLOCATOR) {
        contrl(5) = NONE                # Initialize status to not successful
        i = intin(1)                    # Check locator device for validity
        if (i != DEFAULT & i !=CROSSHAIRS) return

        call gdevot (GS)                # Move to initial position
        call xy40xx (ptsin(1), ptsin(2))
        call gdstot (4, lodcur)
        ginok = OK
        tries = 0
        repeat {
                # enable thumbwheel gin
                call gdstot (4, enable)
                call gdstin (7, line, i)
                if (i <= 5) {  # make sure there are right number of chars
                        for (j=2; j<=i; j=j+1) {
                                # verify chars valid
                                if (line(j) < SPACE | line(j) > QMARK) {
                                        ginok = NONE
                                        call gdstot (2, booboo)
                                        }
                                }
                        }
                else {  # too many chars -- oooops
                        ginok = NONE
                        call gdstot (2, booboo)
                        }
                tries = tries + 1

                } until (ginok == OK | tries > 3)

        # decode the data returned

        intout(1) = NONE

        if (ginok == OK) {

                contrl(5) = 1   # Set successful flag
                contrl(3) = 1   # Set the number of output vertices

                # Return the locator point
                ptsout(1) = mult ((xhi-SPACE), SPACE)+xlo-SPACE
                ptsout(2) = mult ((yhi-SPACE), SPACE)+ylo-SPACE
```

```
                # Return the locator input character
                intout(1) = line(1)
                }
        }

#
#  opcode input string
#
else if (opcode == INPUTxSTRING) {
   if (intin(1) != DEFAULT) {       # Check for valid string device
      contrl(5) = NONE
      return
      }
   ginok = NONE
   i = 0
   itemp = intin(2)                 # Save maximum size
   k = intin(3)                     # Save echo/noecho flag
      repeat {
         if (k == NO) call gdevin (j)   # Get character without echo
         else         call gchrin (j)   # Get character with echo
         if (j == NEWLINE) break
         if (i+1 > itemp) break         # No room in output array
         i = i + 1
         intout(i) = j
         }
   contrl(5) = i                    # Return request status
   }
```

```
#
#  opcode set writing mode
#
else if (opcode == SETxWRITINGxMODE) {
   opcode = intin(1)
   if (opcode == XORxMODE) {          # Device has xor
      j = XORxMODE
      k = 2
      }
   else if (opcode == ERASExMODE) {   # Device has erase
      j = ERASExMODE
      k = 0
      }
   else {                             # Replace mode is default writing mode
      j = REPLACExMODE
      k = 1
      }
   intout(1) = j                 # Return writing mode selected
   ndclrl = k                    # Set appropriate globals to reflect writing
   ndclrm = k                    #    mode
   ndclrf = k
   ndclrt = k
   }

#
#  opcode set input mode
#
else if (opcode == SETxINPUTxMODE) {
   intout(1) = REQUESTxMODE      # Default mode is request
   }

return
end
```

```
subroutine dm40xx (intin)
#########################################################################
#                                                                       #
#          THIS MATERIAL IS CONFIDENTIAL AND IS FURNISHED UNDER         #
#          A WRITTEN LICENSE AGREEMENT.  IT MAY NOT BE USED,            #
#          COPIED OR DISCLOSED TO OTHERS EXCEPT IN ACCORDANCE           #
#          WITH THE TERMS OF THAT AGREEMENT.                            #
#                                                                       #
#          COPYRIGHT (C) 1982 GRAPHIC SOFTWARE SYSTEMS INC.             #
#          ALL RIGHTS RESERVED.                                         #
#                                                                       #
#     Function: Place a marker at the current location on 40xx type     #
#               devices                                                 #
#                                                                       #
#     Input Parameters:                                                 #
#           intin   - x/y location for marker                           #
#     Output Parameters:                                                #
#           none                                                        #
#                                                                       #
#     Routines Called:                                                  #
#           xy40xx - TEK 40xx move/draw routine                         #
#                                                                       #
#########################################################################
define(`MARKxPERIOD',`1')
define(`MARKxPLUS',`2')
define(`MARKxSTAR',`3')
define(`MARKxO',`4')
define(`MARKxX',`5')
define(`FULLSZ',`12')
define(`HALFSZ',`6')
define(`FPERSZ',`4')
define(`HPERSZ',`2')

integer intin(2)

integer fsize, hsize, x1, x2, y1, y2
integer rxy1, rxy2

include(`ddcom')
```

```
   if (ndmktp == MARKxPERIOD) {
      fsize = FPERSZ
      hsize = HPERSZ
      }
   else {
      fsize = FULLSZ
      hsize = HALFSZ
      }

   xl = intin(1) - hsize          #clip marker to device limits
   x2 = xl + fsize
   yl = intin(2) - hsize
   y2 = yl + fsize
   if ((min0(xl,yl) < 0) | (x2 > 1023) | (y2 > 779)) return

   # output appropriate marker centered on location

   call gdevot (GS) # Move to first point
   call xy40xx (xl,yl)

   if (ndmktp == MARKxPERIOD | ndmktp == MARKxO) {
      call xy40xx (x2,yl)
      call xy40xx (x2,y2)
      call xy40xx (xl,y2)
      call xy40xx (xl,yl)
      }
   else {
      if (ndmktp == MARKxX | ndmktp == MARKxSTAR) {
         call xy40xx (x2,y2)
         call gdevot (GS)
         call xy40xx (xl,y2)
         call xy40xx (x2,yl)
         }
      if (ndmktp == MARKxPLUS | ndmktp == MARKxSTAR) {
         call gdevot (GS)
         rxy2 = yl + hsize
         call xy40xx (xl,rxy2)
         call xy40xx (x2,rxy2)
         call gdevot (GS)
         rxyl = xl + hsize
         call xy40xx (rxyl,yl)
         call xy40xx (rxyl,y2)
         }
      }

   return
end
```

```
subroutine dcvret(color)
#########################################################################
#                                                                       #
#         THIS MATERIAL IS CONFIDENTIAL AND IS FURNISED UNDER           #
#         A WRITTEN LICENSE AGREEMENT. IT MAY NOT BE USED,              #
#         COPIED OR DISCLOSED TO OTHERS EXCEPT IN ACCORDANCE            #
#         WITH THE TERMS OF THAT AGREEMENT.                             #
#                                                                       #
#         COPYRIGHT (C) 1982 GRAPHIC SOFTWARE SYSTEMS INC.              #
#         ALL RIGHTS RESERVED                                           #
#                                                                       #
#     Function: Change the color on the retro-graphics terminal         #
#                                                                       #
#     Input Parameters:                                                 #
#           color - color to change to                                  #
#                                                                       #
#     Output Parameters:                                                #
#           none                                                        #
#                                                                       #
#     Routines called:                                                  #
#           gdstot - output a string to the current device              #
#           gimnmx - minmax function                                    #
#                                                                       #
#########################################################################
integer color

SHORTINT i,j
integer fcolor(4), gimnmx

integer ccltb(2), sclrd(2), sclgr(2), sclbl(2), clrflg

include(`ddcom')

common /cmvret/ ccltb, sclrd, sclgr, sclbl, clrflg

#       Set the foreground color
data fcolor/ ESC, SLASH, DIG0, LETD/

if (ndclrp != color | clrflg == YES) { # Does color need to be
                                       # changed
   ndclrp = color             # Set the current color
   clrflg = NO                # reset color flag
```

```
   # Actual color is logical inverse of specified color
   j = gimnmx (color, 0, 1) + 1 #Make sure color index in range
   i = ccltb(j)
   if      (i == 0) j = DIG1   # Use the background color
   else             j = DIG0   # Use the foreground color

   # XOR writing mode takes precedence over color
   # color index 2 is xor
   if (color == 2) j = DIG2

   fcolor(3) = j
   call gdstot (4, fcolor)
   }
return
end
```

```
subroutine xy40xx (kx, ky)
#########################################################################
#                                                                       #
#         THIS MATERIAL IS CONFIDENTIAL AND IS FURNISHED UNDER          #
#         A WRITTEN LICENSE AGREEMENT.  IT MAY NOT BE USED,             #
#         COPIED OR DISCLOSED TO OTHERS EXCEPT IN ACCORDANCE            #
#         WITH THE TERMS OF THAT AGREEMENT.                             #
#                                                                       #
#         COPYRIGHT (C) 1982 GRAPHIC SOFTWARE SYSTEMS INC.              #
#         ALL RIGHTS RESERVED.                                          #
#                                                                       #
#     Function: Convert 40xx x,y coordinate to hiy,loy,hix,lox bytes    #
#               and output them                                         #
#                                                                       #
#     Input Parameters:                                                 #
#          kx       - x-coordinate in 0 to device dependent space     #
#          ky       - y-coorindate in 0 to device dependent space     #
#                                                                       #
#     Output Parameters:                                                #
#          none                                                         #
#                                                                       #
#     Routines Called:                                                  #
#          gdstot - Output a character string to the device             #
#          divid  - divide 2 unsigned 16-bit numbers                    #
#                                                                       #
#########################################################################
# local defines
define(`HIxY',`32')                    #Tek hi y tag
define(`LOxY',`96')                    #Tek lo y tag
define(`HIxX',`32')                    #Tek hi x tag
define(`LOxX',`64')                    #Tek lo x tag
define(`EXTRAxBYTE',`96')              #Tek extra byte tag

integer i, bytes(4), kx,ky, divid

include(`ddcom')

   bytes(1) = divid (ky, 32, i) + HIxY     #shift right 5 bits and set tag
   bytes(2) = i + LOxY                     #set lo bits and add lo tag

   bytes(3) = divid (kx, 32, i) + HIxX     #shift right 5 bits and set tag
   bytes(4) = i + LOxX                     #set lo bits and add lo tag

   call gdstot (4, bytes)

   return
end
```

End of Appendix A

# Appendix B
# VIRTUAL DEVICE INTERFACE SPECIFICATION

---

**INTRODUCTION**

This Appendix contains the specification of the Virtual Device Interface. VDI defines how device drivers interface to GDOS, the device independent portion of GSX-80.

---

**FORMAT**

Function: GSX-80 skeleton device driver

Input Parameters

| | |
|---|---|
| contrl(1) -- | opcode for driver function |
| contrl(2) -- | number of vertices in array ptsin<br>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<br>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 |

Notes

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

All coordinates passed to GSX-80 are in Normalized Device Coordinates (0-32767 along each axis). These units are then mapped to the actual device units (e.g. rasters for CRTs or steps for plotters/printers) by GSX-80 so that all coordinates passed to the device driver are in device units.

Since both input and output coordinates are converted by GSX-80, 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-80. Similarly, the device driver must set contrl(3) to 0 if no x,y coordinates are being returned through GSX-80.

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

The BDOS call to access GSX-80 and the GIOS in CP/M 80 is :

BDOS opcode (in C register) for GSX-80 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 80 this is 2 bytes. For CP/M 86 this is probably 4 bytes.

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

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

---

**OPEN WORKSTATION** Initialize a graphic workstation

---

**Input**

| | | |
|---|---|---|
| contrl(1) | -- | Opcode = 1 |
| contrl(2) | -- | 0 |
| contrl(4) | -- | Length of intin = 10 |
| intin | -- | Initial defaults (line style, color, character size, etc) |
| intin(1) | -- | Workstation identifier (i.e. device driver id) This value is used to determine which device driver to dynamically load into memory. |
| intin(2) | -- | Line type |
| 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 (e.g. 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 (e.g. 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 (crt's)
intout(4) -- Width of one pixel (plotter step...) in micrometers
intout(5) -- Height of one pixel (plotter step...) in micrometers
intout(6) -- Number of character heights (0 = continuous scaling)
intout(7) -- Number of line types
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 pre-defined 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 GDPs

```
intout(16)-intout(25) -- list of GDPs (up to
                         10 allowed)
                        -1  -- GDP does not
                               exist
intout(26)-intout(35) -- attribute set asso-
                         ciated with each GDP
                        -1  -- GDP does not
                               exist
                         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)--   Pixel operation capability
               flag
               0 -- no
               1 -- yes
intout(40)--   Number of available colors
               (total number of colors in
               color palette)
               0 -- continuous device
               2 -- monochrome (black and
                    white)
              >2 -- number of colors avail-
                    able
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<br>0 -- Output only<br>1 -- Input only<br>2 -- Input/Output<br>3 -- Device independent segment storage<br>4 -- GKS Metafile output |
| ptsout(1) -- | 0 |
| ptsout(2) -- | Minimum character height in device units |
| ptsout(3) -- | 0 |
| ptsout(4) -- | Maximum character height in device units |
| 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 |
| ptsout(11)-- | 0 |
| ptsout(12)-- | Maximum marker height in device units |

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

Monochrome

| Index | Color |
|---|---|
| 0 | Black |
| 1 | White |

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:

Character height = minimumcharacter 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.

---

**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.

---

**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 and hardcopy devices to perform a top-of-form operation. On plotters without paper advance, the operator is prompted to load a new page.

---

**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 which are queued to be executed immediately. (Analogous to flushing buffers.)

---

**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 = 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
13= REVERSE VIDEO ON
14= REVERSE VIDEO OFF

15= INQUIRE CURRENT CURSOR ADDRESS
16= INQUIRE TABLET STATUS
17= HARDCOPY
18= PLACE CURSOR AT LOCATION
19= REMOVE CURSOR
20-50= UNUSED BUT RESERVED FOR FUTURE EXPANSION
51-100= UNUSED AND AVAILABLE FOR USE

| | | | |
|---|---|---|---|
| | intin | -- | function dependent information |
| | 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 applications program. Some escape functions are pre-defined above, but others can be defined for your particular devices. The parameters passed are dependent on the function being performed.

---

**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.

---

**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. This is used to explicitly exit alpha cursor addressing mode.

---

**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. This is used to explicitly enter the alpha cursor addressing mode.

---

**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.

---

**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.

---

**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

---

**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.

---

**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).

---

**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.





3C

---

**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.

---

**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.

---

**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 Decimal Equivalent |
| **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).

---

**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 rendered in reverse video format, that is, characters are dark on a light background.

---

**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 rendered in normal video format, that is, characters are light on a dark background.

---

**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.

---

**INQUIRE TABLET STATUS** Return tablet status

---

| | | |
|---|---|---|
| **Input** | contrl(2) -- | 0 |
| | contrl(6) -- | function id = 16 |
| **Output** | contrl(3) -- | 0 |
| | intout(1) -- | tablet status<br>0 = tablet not available<br>1 = tablet available |

**Description** This operation indicates whether a graphics tablet is connected to the workstation.

---

**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 may entail copying the screen to a printer or other attached hardcopy device.

---

**PLACE CURSOR AT LOCATION** Place a 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 cursor/marker at the specified location. This is device dependent and can be an underbar, block, etc.

---

**REMOVE CURSOR** Remove cursor/marker

---

**Input**

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

**Output**

contrl(3) -- 0

**Description**

This operation makes the cursor invisible on the screen.

---

**POLYLINE** Outputa polyline 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 (rasters, plotter steps, etc.)
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, line type, line width.

---

**POLYMARKER** Output markers to the device

---

**Input**

```
contrl(1) --  opcode = 7
contrl(2) --  number of markers
ptsin     --  array of coordinates in device
              units (n) (rasters, plotter
              steps, etc.)
              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. Be sure to specify the solid line style before drawing markers, and restore the previous line style when done. Also, make sure the markers exhibit the current marker attributes: color, scale, 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 | -- | character string in ASCII Decimal Equivalent |
| 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.

---

**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. Make sure the correct color, fill interior style 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 insure that the fill area is closed by connecting the first point to the last point.

---

**CELL ARRAY** Define 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
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
0 -- clear
1 -- set
2 -- or
3 -- and
4 -- complement (xor)
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 can't do cell arrays it must at least outline the area in the current line color.

---

**GENERALIZED DRAWING PRIMITIVE** 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, line type, 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 of GDP 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 |
| 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 (Diablo, Epson, etc.) |
| | 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 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 pre-defined and others are available for expansion.

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

---

**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 character cell selected. This is a best fit match to the requested character size.

---

**SET CHARACTER UP VECTOR** Set text direction

---

**Input**

contrl(1) -- opcode = 13
contrl(2) -- 0
intin(1) -- requested angle of rotation (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
intout(1) -- angle of rotation 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 which is a best fit match to the requested value.

---

**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 indices are required (black and white for monochrome).

---

**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 4 linestyles are required:

1 - solid
2 - dashed
3 - dotted
4 - dashed- dotted

If the requested linestyle is out of range then line style 1 (solid) should be used.

---

**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. The width is specified in DC.

---

**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 indices are required.

---

**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 5 marker types are required :

```
1 - .
2 - +
3 - *
4 - O
5 - X
```

If the requested marker type is out of range then type 3 should be used.

---

**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 which is a best fit to the requested 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 indices are required.

---

**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.

---

**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 2 color indices are required.

---

**SET FILL INTERIOR STYLE** Set interior fill style

---

**Input**

```
contrl(1) --   opcode = 23
contrl(2) --   0
intin(1)  --   requested fill interior style
               0 - Hollow
               1 - Solid
               2 - 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, then Hollow should be used. 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. If the requested index is not available then index 1 should be used. The index references a hatch style (+45 degrees or -45 degrees) if the fill interior style is hatch, or it references a pattern (stars, dots, etc.) if the interior fill style is pattern.

---

**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 2 color indices are required.

---

**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.

---

**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<br>0 -- if no errors<br>1 -- if a color value could not be determined for some pixel |
| | intout -- | color index array (stored one row at time)<br>-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. Note that the upper and lower y-coordinates are identical since only one row is returned at a time. The returned array is the sequence of color indices across the specified row from left to right.

NOTE: Color indices are returned one row at a time, starting from the top of the rectangular area, proceeding downward.





4A

---

**INPUT LOCATOR** Return locator position

---

**For REQUEST MODE Input:**

| | | | |
|---|---|---|---|
| **Input** | contrl(1) | -- | opcode = 28 |
| | contrl(2) | -- | number of input vertices = 1 |
| | intin(1) | -- | locator device number<br>1 = default locator device<br>2 = crosshairs<br>3 = graphics tablet<br>4 = joystick<br>5 = lightpen<br>6 = plotter<br>7 = mouse<br>8 = trackball<br>>8 = workstation dependent |
| | 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<br>0 = request unsuccessful<br>>0 = request successful |
| | intout(1) | -- | locator terminator<br>For keyboard terminated locator input, this is the ASCII Decimal Equivalent (ADE) of the key struck to terminate input. For non-keyboard terminated input (tablet, mouse, etc.), valid locator terminators begin with <space> (ADE 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 <!> (ADE 33), the third a <"> (ADE 34), and the fourth a <#> (ADE 35).

ptsout(1) -- final x-coordinate of locator in device units

ptsout(2) -- final y-coordinate of locator in device units

**For SAMPLE MODE Input:**

**Input**

contrl(1) -- opcode = 28

contrl(2) -- number of input vertices = 0

intin(1) -- locator device number
1 = default locator device
2 = crosshairs
3 = graphics tablet
4 = joystick
5 = lightpen
6 = plotter
7 = mouse
8 = trackball
>8 = workstation dependent

**Output**

contrl(3) -- number of output vertices
1 = sample successful
0 = sample unsuccessful

contrl(5) -- length of intout array -- status
0 = sample unsuccessful
>0 = sample successful

ptsout(1) -- current x-coordinate of locator in device units

ptsout(2) -- current y-coordinate of locator in device units

**Description**

This operation returns the position in DC coordinates of the specified locator device.





4C

---

**INPUT VALUATOR** Return value of valuator device

---

**For REQUEST MODE Input:**

| | | | |
|---|---|---|---|
| **Input** | contrl(1) | -- | opcode = 29 |
| | contrl(2) | -- | 0 |
| | intin(1) | -- | valuator device number<br>1 -- default valuator device |
| | intin(2) | -- | initial value |
| **Output** | contrl(3) | -- | 0 |
| | contrl(5) | -- | length of intout array --<br>status<br>0 = request unsuccessful<br>>0 = request successful |
| | intout(1) | -- | output value |

**For SAMPLE MODE Input:**

| | | | |
|---|---|---|---|
| **Input** | contrl(1) | -- | opcode = 29 |
| | contrl(2) | -- | 0 |
| | intin(1) | -- | valuator device number<br>1 -- default valuator device |
| **Output** | contrl(3) | -- | 0 |
| | contrl(5) | -- | length of intout array --<br>status<br>0 = sample unsuccessful<br>>0 = sample successful |
| | intout(1) | -- | current valuator value if sample successful |

**Description** This operation returns the current value of the valuator device.

---

**INPUT CHOICE** Return choice device status

---

**For REQUEST MODE Input:**

| | | | |
|---|---|---|---|
| **Input** | contrl(1) | -- | opcode = 30 |
| | contrl(2) | -- | 0 |
| | intin(1) | -- | choice device number<br>1 = default choice device<br>2 = function key<br>>2 = workstation dependent |
| | intin(2) | -- | initial choice number |
| **Output** | contrl(3) | -- | 0 |
| | contrl(5) | -- | Length of intout array -- status<br>0 = request unsuccessful<br>>0 = request successful |
| | intout(1) | -- | choice number |

**For SAMPLE MODE Input:**

| | | | |
|---|---|---|---|
| **Input** | contrl(1) | -- | opcode = 30 |
| | contrl(2) | -- | 0 |
| | intin(1) | -- | choice device number<br>1 = default choice device<br>2 = function key<br>>2 = workstation dependent |
| **Output** | contrl(3) | -- | 0 |
| | contrl(5) | -- | Length of intout array -- status<br>0 = sample unsuccessful<br>>0 = sample successful |
| | intout(1) | -- | choice number or 0 if sample unsuccessful |

**Description** This operation returns the choice status of the specified choice device. The range of choice numbers is device dependent.

---

**INPUT STRING** Return string from specified string device

---

**For REQUEST MODE Input:**

| | | | |
|---|---|---|---|
| **Input** | contrl(1) | -- | opcode = 31 |
| | contrl(2) | -- | 0 |
| | intin(1) | -- | string device number<br>1 = default string device (keyboard) |
| | intin(2) | -- | maximum string length |
| | intin(3) | -- | echo mode<br>0 = don't echo input characters<br>1 = echo input characters |
| **Output** | contrl(3) | -- | 0 |
| | contrl(5) | -- | length of output string<br>0 = request unsuccessful<br>>0 = request successful |
| | intout | -- | output string |

---

**For SAMPLE MODE Input:**

| | | |
|---|---|---|
| **Input** | contrl(1) -- | opcode = 31 |
| | contrl(2) -- | 0 |
| | intin(1) -- | string device number<br>1 = default string device (keyboard) |
| | intin(2) -- | maximum string length |
| | intin(3) -- | echo mode<br>0 = don't echo input characters<br>1 = echo input characters |
| **Output** | contrl(3) -- | 0 |
| | contrl(5) -- | length of output string<br>0 = sample unsuccessful (characters not available)<br>>0 = sample successful (characters available) |
| | intout -- | output string if sample successful |

**Description** The Request String operation requests a string from the specified device. The default device is the keyboard.





4B

---

**SET WRITING MODE** Set writing mode

---

**Input**

```
contrl(1) --   opcode = 32
contrl(2) --   0
intin(1)  --   writing mode
               1 = replace
               2 = overstrike
               3 = complement(xor)
               4 = erase
```

**Output**

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

**Description**

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

---

**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    --   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/location of the input device without waiting.

**REQUIRED OPCODE CRT DEVICES**

The following opcodes (and sub-functions) are required 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 |
| 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 |

| | |
|---|---|
| 6 | Polyline |
| 7 | Polymarker |
| 8 | Text |
| 9 | Filled area |
| 10 | Cell array |
| 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 PLOTTERS/PRINTERS**

The following opcodes (and sub-functions) are required for plotters / printers:

| 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 |
| 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) |

Determining if a non-required opcode is available in a particular driver may 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 B

# Appendix C
# GLOSSARY

**Assignment Table** — The Assignment Table associates logical device numbers, called workstation IDs, with specific device driver files so that devices may be referred to by number within the application program. The Assignment Table resides in a text file called ASSIGN.SYS and may be modified using any text editor.

**BDOS** — BDOS is the CP/M Basic Disk Operating System. It contains the device independent portion of the CP/M file control system. The device dependent parts of standard CP/M are found in the BIOS (Basic I/O System) module.

**COM file** — A .COM extension to a filename is reserved for executable program files.

**Coordinate scaling** — Coordinate scaling transforms points from one "space" to another. In GSX-80 all point coordinates must be specified in Normalized Device Coordinates with values between 0 and 32,767. GDOS will then scale these coordinates into values which are appropriate for your graphics device.

**Default device driver** — The largest driver loaded during a graphics session. It is always the first driver named in the Assignment Table.

**Device driver** — A device driver translates between the standard, device-independent portion of an operating system and the specific command sequences for a particular device. Device drivers for graphics devices are contained in the GIOS (Graphics I/O System) portion of GSX-80.

---

**Function code** A function code is a number which indicates to the operating system what function is being requested when a service call is made. All graphics functions use function code 115. The particular graphics operation desired is specified by an operation code in the parameter list passed to GDOS.

**GDOS** The Graphics Device Operating System, or GDOS, is the device independent portion of GSX-80. It services graphics requests and calls GIOS to send commands to graphics devices.

**GENGRAF** GENGRAF is a special utility which permanently attaches the GSX Loader to your application program.

**GIOS** The Graphics Input Output System, or GIOS, is the device dependent portion of GSX-80. GIOS refers to the individual device drivers which translate between a particular device and the standard VDI conventions.

**GKS** Abbrevation for Graphical Kernel System.

**GSS-KERNEL** GSS-KERNEL is a graphics utility package from Graphic Software Systems, Inc. which provides a Graphical Kernel System (GKS) interface to the programmer. GSS-KERNEL employs GSX-80 to interface to the graphics devices on your system.

**GSS-PLOT** GSS-PLOT is a graphics application package from Graphic Software Systems, Inc. which allows you to create graphs and charts using high level procedure calls.

**GSX-80** The Graphics System Extension, or GSX-80, is the graphics extension to the CP/M family of operating systems.

| | |
|---|---|
| **GSX Loader** | The GSX Loader is a special program which is attached by the GENGRAF utility to the front of graphics application programs run under GSX-80. The GSX Loader brings GSX-80 into memory when a graphics application is executed and sets up the CP/M environment for GSX. |
| **Graphical Kernel System** | The Graphical Kernel System (GKS) is an international standard for the programmer's interface to graphics. |
| **Graphics primitives** | Graphics primitives are the basic graphics operations performed by GSX-80; for example, drawing lines, markers and text strings. |
| **NDC** | Abbreviation for Normalized Device Coordinates. |
| **Normalized Device Coordinate Space** | Normalized Device Coordinate Space is a uniform virtual space by which a graphics application program passes graphics information to a device. GDOS translates between NDC space and the display coordinates of a particular device. |
| **Normalized Device Coordinates** | The Normalized Device Coordinate (NDC) Space is a 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** | An operation code is passed to GDOS as part of a parameter list and indicates which graphics operation is requested. |
| **PRL file** | The .PRL extension to a filename is reserved for "page relocatable modules," that is, modules which may be loaded into different locations in memory at run time. All GSX-80 device drivers must be in PRL format since they are loaded dynamically at run time by the GSX Loader or GDOS. |

| | |
|---|---|
| **TPA** | Abbreviation for Transient Program Area. |
| **Transient Program Area** | The Transient Program Area is the CP/M nomenclature for the memory area available for user application programs. |
| **VDI** | Abbreviation for Virtual Device Interface. |
| **Virtual Device Interface** | The Virtual Device Interface is a 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-80 is based on VDI and all device drivers written for GSX-80 must conform to the VDI specification. |
| **Workstation** | A workstation is a graphics device with one display surface and zero or more input devices. |
| **Workstation Identification Number (ID)** | A workstation ID is a logical unit number which 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 Appendix C

# Appendix D
# DEVICE SPECIFICS

This Appendix contains specific information about the devices supported by GSX-80.

## EPSON MX-80 PRINTER WITH GRAFTRAX PLUS

| | |
|---|---|
| **FILE NAME** | DDMX80.PRL |
| **DEVICE INDEX** | The actual device index for this device is determined in the ASSIGN.SYS file, which associates a device index with a GIOS module (device driver). For printers, this index must be in the range 21-30. |
| **MAXIMUM BAUD RATE** | 9600 baud |
| **COMMUNICATIONS** | Standard serial communications (RS-232C). |
| **GRAPHIC INPUT (GIN)** | The device does not support graphic input. |
| **TEXT** | The printer supports six character sizes. Text can be rotated in 90 degree increments. |
| **MARKERS** | 1 .<br>2 +<br>3 *<br>4 O<br>5 X |
| **LINESTYLE** | The printer has five hardware line styles. Line style 1 is solid and line styles 2 - 5 are combinations of dashed and dotted lines. |
| **COLOR** | The MX-80 printer supports two colors. Index 1 is displayed with the black ribbon and index 0 is not displayed. These colors can not be redefined. |
| **GENERALIZED DRAWING PRIMITIVES (GDPs)** | No GDPs are available on the Epson MX-80. |

**ESCAPES**

The escape functions available on the MX-80 printer are:

1 Inquire addressable character cells

**SUMMARY**

The functions available in the MX-80 printer GIOS are:

| Opcode | Definition |
|---|---|
| 1 | Open workstation |
| 2 | Close workstation |
| 3 | Clear workstation |
| 4 | Update workstation |
| 5 | Escape |
| 6 | Polyline |
| 7 | Polymarker |
| 8 | Text |
| 9 | Filled area |
| 12 | Set character height |
| 13 | Set character up vector |
| 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 |

REFER TO DEVICE DOCUMENTATION if you have other questions regarding this particular device.

## HEWLETT-PACKARD 7220 GRAPHICS PLOTTER

**FILENAME** DD7220.PRL

**DEVICE INDEX** The actual device index for this device is determined in the ASSIGN.SYS file, which associates a device index with a GIOS module (device driver). For plotters, this index must be in the range 11-20.

**MAXIMUM BAUD RATE** 2400 baud

**COMMUNICATIONS** Standard serial communications (RS-232C).

**GRAPHIC INPUT (GIN)** The pen holder is used to indicate what point is to be input. The pen holder is moved by pressing the position keys on the front panel. When the cursor is at the desired location, the point can be selected by pressing the ENTER button. This causes the coordinates of the point to be transmitted back to the user program.

**TEXT** The HP 7220 has continuous scaling of character sizes. Text can be rotated in one degree increments.

**MARKERS**

| | |
|---|---|
| 1 | . |
| 2 | + |
| 3 | * |
| 4 | O |
| 5 | X |

**LINESTYLE** The 7220 plotter has seven hardware line styles. Line style 1 is solid and line styles (2-7) are combinations of dashed and dotted lines.

**COLOR**

The 7220 has eight pens. The index parameter in the routines that set a color index correspond directly to a plotter pen number (i.e. index 0 corresponds to pen 1, index 1 to pen 2, ... index 7 to pen 8). Indices greater than 7 are mapped to pen 8. Indices less than 0 are mapped to pen 1.

**GENERALIZED DRAWING PRIMITIVES (GDPs)**

No GDPs are available on the HP7220.

**ESCAPES**

The escape functions available on this device are :

1 Inquire addressable character cells

**SUMMARY**

The functions available in the HP7220 GIOS are:

| Opcode | Definition |
|---|---|
| 1 | Open workstation |
| 2 | Close workstation |
| 3 | Clear workstation |
| 4 | Update workstation |
| 5 | Escape |
| 6 | Polyline |
| 7 | Polymarker |
| 8 | Text |
| 9 | Filled area |
| 10 | Cell array |
| 12 | Set character height |
| 13 | Set character up vector |
| 14 | Set color representation |
| 15 | Set polyline linetype |
| 17 | Set polyline color index |
| 18 | Set polymarker type |
| 20 | Set polymarker color index |
| 21 | Set text font |
| 22 | Set text color index |
| 25 | Set fill color index |

| | |
|---|---|
| 26 | Inquire color representation |
| 28 | Input locator |
| 33 | Set input mode - request only |

REFER TO DEVICE DOCUMENTATION if you have other questions regarding this particular device.

## HEWLETT-PACKARD 7470A GRAPHICS PLOTTER

**FILENAME** DD7470.PRL

**DEVICE INDEX** The actual device index for this device is determined in the ASSIGN.SYS file, which associates a device index with a GIOS module (device driver). For plotters, this index must be in the range 11-20.

**MAXIMUM BAUD RATE** 9600 baud

**COMMUNICATIONS** Standard serial communications (RS-232C).

**GRAPHIC INPUT (GIN)** The pen holder is used to indicate what point is to be input. The pen holder is moved by pressing the position keys on the front panel. When the cursor is at the desired location, the point can be selected by pressing the ENTER button. This causes the coordinates of the point to be transmitted back to the user program.

**TEXT** The HP 7470A has continuous scaling of character sizes. Text can be rotated in one degree increments.

**MARKERS**

1 .
2 +
3 *
4 O
5 X

**LINESTYLE** The 7470A plotter has seven hardware line styles. Line style 1 is solid and line styles (2-7) are combinations of dashed and dotted lines.

**COLOR**

Colors are refered to on the 7470A by the number of the pen and not by the pen holder. This gives the flexibility of more than two colors on the plotter. By default, index 1 is held in pen holder 1 and index 2 is held in pen holder 2. If the user is using more than these two colors, then a prompt will be generated, telling the user to insert the desired color in a pen station and then enter the pen station. So, the index parameter in the routines that refer to a color index corresponds to a pen number not a pen holder (i.e. index 1 corresponds to pen 1, index 2 to pen 2, index 3 to pen 3...). There is no limit to the number of pen indices available on the plotter.

**GENERALIZED DRAWING PRIMITIVES (GDPs)**

No GDPs are available on the HP7470A.

**ESCAPES**

The escape functions available on this device are :

1 Inquire addressable character cells

**SUMMARY**

The functions available in the HP7470 GIOS are:

| Opcode | Definition |
|---|---|
| 1 | Open workstation |
| 2 | Close workstation |
| 3 | Clear workstation |
| 4 | Update workstation |
| 5 | Escape |
| 6 | Polyline |
| 7 | Polymarker |
| 8 | Text |
| 9 | Filled area |
| 10 | Cell array |
| 12 | Set character height |
| 13 | Set character up vector |
| 14 | Set color representation |

| | |
|---|---|
| 15 | Set polyline linetype |
| 17 | Set polyline color index |
| 18 | Set polymarker type |
| 19 | Set polymarker scale |
| 20 | Set polymarker color index |
| 21 | Set text font |
| 22 | Set text color index |
| 25 | Set fill color index |
| 26 | Inquire color representation |
| 28 | Input locator |
| 33 | Set input mode - request only |

REFER TO DEVICE DOCUMENTATION if you have other questions regarding this particular device.

## DIGITAL ENGINEERING RETRO-GRAPHICS (GEN.II)

| | |
|---|---|
| **FILE NAME** | DDGEN2.PRL |
| **DEVICE INDEX** | The actual device index for this device is determined in the ASSIGN.SYS file, which associates a device index with a GIOS module (device driver). For crts, this index must be in the range 1-10. |
| **MAXIMUM BAUD RATE** | 9600 baud for all graphics |
| **COMMUNICATIONS** | Standard serial communications (RS-232C). The GEN.II uses status flagging via the "!REP 0" command to avoid losing data at high baud rates. |
| **GRAPHIC INPUT (GIN)** | When GIN is invoked on the GEN.II Retro-Graphics terminal, a crosshair cursor appears on the screen. The crosshair can be moved by pressing one of the four arrow keys (up, down, left, right) on the keyboard. When the cursor is at the desired location, the point can be selected by pressing any alphanumeric key (other than RETURN) on the keyboard. This causes the coordinates of the point to be transmitted back to the user program. |
| **TEXT** | The GEN.II has continuous scaling character sizes. Text can be rotated in one-degree increments. Two fonts, standard ASCII vector characters and user-defined vector characters, are available. |
| **MARKERS** | 1 .<br>2 +<br>3 *<br>4 O<br>5 X |

**LINESTYLE**

The GEN.II has eight hardware line styles. Line style 1 is solid and line styles 2 - 8 are combinations of dashed and dotted lines.

**COLOR**

The GEN.II Retro-Graphics enhancement supports both color and monochrome terminals. On monochrome terminals, color specifications are mapped to appropriate dithering patterns, but all lines and borders are drawn in either white or black. Areas may be filled with any of 8 color indices or 120 dithering patterns. On color devices, color indices are mapped to one of 8 colors. The default association of color indices for both monochrome and color devices is:

| Index | Monochrome | Color |
|---|---|---|
| 0 | dithering pattern 0 | Black |
| 1 | dithering pattern 3 | Red |
| 2 | dithering pattern 12 | Green |
| 3 | dithering pattern 48 | Blue |
| 4 | dithering pattern 15 | Cyan |
| 5 | dithering pattern 16 | Yellow |
| 6 | dithering pattern 51 | Magenta |
| 7 | dithering pattern 63 | White |

**GENERALIZED DRAWING PRIMITIVES (GDPs)**

The available GDPs and their identifiers are:

1 - Bars
2 - Arcs
3 - Pie Slices
4 - Circles

**ESCAPES**

The escape functions available on the GEN.II terminal are:

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 |
| 11 | Direct cursor address |
| 12 | Output cursor addressable text |
| 13 | Reverse video on |
| 14 | Reverse video off |
| 15 | Inquire current cursor address |
| 17 | Hardcopy |

**SUMMARY**

The functions available in the GEN.II Retro-Graphics GIOS are:

| Opcode | Definition |
|---|---|
| 1 | Open workstation |
| 2 | Close workstation |
| 3 | Clear workstation |
| 4 | Update workstation |
| 5 | Escape |
| 6 | Polyline |
| 7 | Polymarker |
| 8 | Text |
| 9 | Filled area |
| 10 | Cell array |
| 11 | Generalized Drawing Primitives |
| 12 | Set character height |
| 13 | Set character up vector |
| 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 |
| 23 | Set fill interior style |
| 24 | Set fill style index |
| 25 | Set fill color index |
| 26 | Inquire color representation |
| 27 | Inquire cell array |
| 28 | Input locator |

| | |
|---|---|
| 31 | Input string |
| 32 | Set writing mode - replace, xor, erase |
| 33 | Set input mode - request |

REFER TO DEVICE DOCUMENTATION if you have other questions regarding this particular device.





5A

## HOUSTON INSTRUMENTS HIPLOT DMP-3/4-443 MULTIPEN PLOTTER

**FILENAME** DDHI3M.PRL

**DEVICE INDEX** The actual device index for this device is determined in the ASSIGN.SYS file, which associates a device index with a GIOS module (device driver). For plotters, this index must be in the range 11-20.

**MAXIMUM BAUD RATE** 9600 baud

**COMMUNICATIONS** Standard serial communications (RS-232C). The Clear to Send (CTS) signal must be functional at the host and carried through to pin 5 at the plotter connector. Also pin 9 must be jumpered to pin 7 at the plotter connector to enable HIPLOT mode 2 communications. The Input/Output uses this form of handshaking communications since, at high baud rates, the plotter can not plot data as fast as it receives data from the computer. Also note that to set the baud rate at the plotter end, pin 6 must be wired to one of the following pins:

| Pin | Baud Rate |
|---|---|
| 14 | 9600 |
| 15 | 4800 |
| 16 | 2400 |
| 17 | 1200 |
| 18 | 600 |
| 19 | 300 |

**GRAPHIC INPUT (GIN)** The plotter does not support GIN.

**TEXT** The DMP-3/4-443 has five character sizes. Text can be rotated in 90 degree increments.

**MARKERS** See sample below.

---

**LINESTYLE**

The DMP-3/4-443 Multipen plotter has nine hardware line styles. Line style 1 is solid and line styles (2-9) are combinations of dashed and dotted lines.

**COLOR**

The DMP-3/4-443 has six pens. The index parameter in the routines that set a color index correspond directly to a plotter pen number (i.e. index 0 corresponds to pen 1, index 1 to pen 2, ... index 5 to pen 6). Indices greater than 5 are mapped to pen 6. Indices less than 0 are mapped to pen 1.

**GENERALIZED DRAWING PRIMITIVES (GDPs)**

No GDPs are available on the DMP-3/4-443.

**ESCAPES**

The escape functions available on this device are :

1 Inquire addressable character cells

**SUMMARY**

The functions available in the DMP-3/4 GIOS are:

| Opcode | Definition |
|---|---|
| 1 | Open workstation |
| 2 | Close workstation |
| 3 | Clear workstation |
| 4 | Update workstation |
| 5 | Escape |
| 6 | Polyline |
| 7 | Polymarker |
| 8 | Text |
| 9 | Filled area |
| 10 | Cell array |
| 12 | Set character height |
| 13 | Set character up vector |
| 14 | Set color representation |
| 15 | Set polyline linetype |
| 17 | Set polyline color index |
| 18 | Set polymarker type |
| 19 | Set polymarker scale |





5C

| | |
|---|---|
| 20 | Set polymarker color index |
| 22 | Set text color index |
| 25 | Set fill color index |
| 26 | Inquire color representation |

REFER TO DEVICE DOCUMENTATION if you have other questions regarding this particular device.

## HOUSTON INSTRUMENTS HIPLOT DMP-6/7 MULTIPEN PLOTTER

| | |
|---|---|
| **FILENAME** | DDHI7M.PRL |
| **DEVICE INDEX** | The actual device index for this device is determined in the ASSIGN.SYS file, which associates a device index with a GIOS module (device driver). For plotters, this index must be in the range 11-20. |
| **MAXIMUM BAUD RATE** | 9600 baud |
| **COMMUNICATIONS** | Standard serial communications (RS-232C). |
| **GRAPHIC INPUT (GIN)** | The plotter does not support GIN. |
| **TEXT** | The DMP-6/7 has nine character sizes. Text can be rotated in 90 degree increments. |
| **MARKERS** | See sample below. |
| **LINESTYLE** | The DMP-6/7 Multipen plotter has nine hardware line styles. Line style 1 is solid and line styles (2-9) are combinations of dashed and dotted lines. |
| **COLOR** | The DMP-6/7 has eight pens. The index parameter in the routines that set a color index correspond directly to a plotter pen number (i.e. index 0 corresponds to pen 1, index 1 to pen 2, ... index 7 to pen 8). Indices greater than 7 are mapped to pen 8. Indices less than 0 are mapped to pen 1. |
| **GENERALIZED DRAWING PRIMITIVES (GDPs)** | The GDPs available on the DMP-6/7 are:<br>2 Arc |
| **ESCAPES** | The escape functions available on this device are :<br>1 Inquire addressable character cells |

---

**SUMMARY**

The functions available in the DMP-6/7 GIOS are:

| Opcode | Definition |
|---|---|
| 1 | Open workstation |
| 2 | Close workstation |
| 3 | Clear workstation |
| 4 | Update workstation |
| 5 | Escape |
| 6 | Polyline |
| 7 | Polymarker |
| 8 | Text |
| 9 | Filled area |
| 10 | Cell array |
| 11 | Generalized drawing primitive |
| 12 | Set character height |
| 13 | Set character up vector |
| 14 | Set color representation |
| 15 | Set polyline linetype |
| 17 | Set polyline color index |
| 18 | Set polymarker type |
| 19 | Set polymarker scale |
| 20 | Set polymarker color index |
| 22 | Set text color index |
| 25 | Set fill color index |
| 26 | Inquire color representation |

REFER TO DEVICE DOCUMENTATION if you have other questions regarding this particular device.

## VT100 WITH DIGITAL ENGINEERING RETROGRAPHICS

| | |
|---|---|
| **FILENAME** | DDVRET.PRL |
| **DEVICE INDEX** | The actual device index for this device is determined in the ASSIGN.SYS file, which associates a device index with a GIOS module (device driver). For crts, this index must be in the range 1-10. |
| **MAXIMUM BAUD RATE** | 9600 baud for all graphics |
| **COMMUNICATIONS** | Standard serial communications (RS-232C). The VT100 uses XON / XOFF flagging to avoid losing data at high baud rates. |
| **GRAPHIC INPUT (GIN)** | When GIN is invoked on the VT100 Retrographics terminal, a crosshair cursor appears on the screen. The crosshair can be moved by pressing one on the four arrow keys (up, down, left, right) on the top row of keys on the keyboard. When the cursor is at the desired location, the point can be selected by pressing any alphanumeric key (other than return) on the keyboard. This causes the coordinates of the point to be transmitted back to the user program. The terminal must be set up so that GIN is terminated by CR only. This can be done by setting the two trailer codes in Retrographics set-up mode to 0D hex and FF hex respectively. Refer to the instructions in the User Manual for Retrographics Model VT640, "Set Up Procedures," for a further discussion of trailer characters. |
| **TEXT** | The VT100 has four character sizes. It cannot rotate text. |

**MARKERS**

1 .
2 +
3 *
4 O
5 X

**LINESTYLE**

The VT100 has five hardware line styles. Line style 1 is solid and line styles 2 - 5 are combinations of dashed and dotted lines.

**COLOR**

The VT100 is a monochrome terminal with only two levels of gray scale / intensity (black and white). Color specifications are mapped to an appropriate gray scale / intensity. All colors other than black are mapped to white.

The default association of color indices with gray scale / monochrome intensity is :

0 0% Intensity - Black
1 100% Intensity - White

**GENERALIZED DRAWING PRIMITIVES (GDPs)**

No GDPs are available on the VT100.

**ESCAPES**

The escape functions available on this terminal are :

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
13 Reverse video on

14 Reverse video off
15 Inquire current cursor address

**SUMMARY**

The functions available in the VT100 Retrographics GIOS are:

| Opcode | Definition |
|---|---|
| 1 | Open workstation |
| 2 | Close workstation |
| 3 | Clear workstation |
| 4 | Update workstation |
| 5 | Escape |
| 6 | Polyline |
| 7 | Polymarker |
| 8 | Text |
| 9 | Filled area |
| 10 | Cell array |
| 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 |
| 28 | Input locator |
| 31 | Input string |
| 32 | Set writing mode - replace, xor, erase |
| 33 | Set input mode - request |

REFER TO DEVICE DOCUMENTATION if you have other questions regarding this particular device.

End of Appendix D

# INDEX