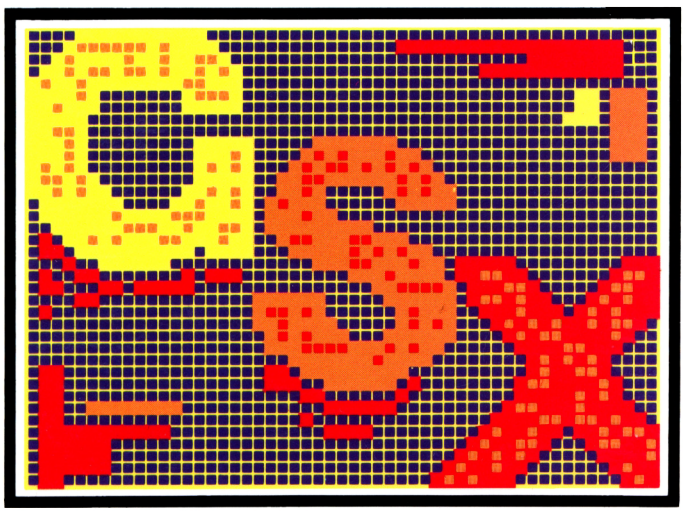
Digital Research Inc.

INCORPORATING USER'S MANUAL · PROGRAMMER'S MANUAL · LANGUAGE REFERENCE MANUAL

GSX HANDBOOK



Digital Research Inc.

GLENTOP
PUBLISHERS ■ LIMITED

GSX HANDBOOK

GLENTOP PUBLISHERS LTD

ISBN 1 85181 056 0

Printed by the Wembley Press Ltd., Reading and London

DIGITAL RESEARCH

GSX-86
Graphics Extension
User's Guide

****

Second Edition: March 1984

****

Foreword

INTRODUCTION

This *GSX-86 Graphics Extension User's Guide* explains the features of GSX-86, Graphics System Extension for microcomputer operating systems. GSX-86 lets you use graphic applicatons an many types of printers, plotters and graphics cards. GSX-86 supports the following operating systems:

- CP/M family
- MS -DOS
- PC DOS

If you are a new user of GSX-86, this guide helps you install GSX-86 on your microcomputer system so that you can use your graphics applications with the many devices GSX-86 supports.

GSX-86

GSX-86 adds graphic capability and a device-independent operating environment to supported operating systems. After you install GSX-86, your system has the following features:

- You can use many types of plotters, printers, and other gaphics devices.
- You can use applications that use GSX-86 on many types of microcomputers.
- You can use graphic applicatons such as DR Graph and DR Draw to create high-quality presentation and creative graphics. DR Graph allows you to graph and plot data by making simple menu selections. DR Draw allows you to draw complex graphic designs with your microcomputer.

HOW THIS GUIDE IS ORGANIZED

This guide is organized in four sections. Section 1 describes how GSX-86 works. Section 2 tells you how to start GSX-86. Sections 3 and 4 describe and give information on using GINSTALL, the GSX-86 installation program.

CONVENTIONS

This guide uses several conventions:

- For clarity, commands and keystrokes you enter are capitalized and appear in boldface type. However, you do not need to capitalize commands.

- The CONTROL key is represented by the symbol ^. This symbol followed by an alphabetic character means that you must press the CONTROL key and the alphabetic character key simultaneously. For example, ^W means that you must press the CONTROL key and the W key simultaneously.

Table of Contents

Table of Contents (continued)

APPENDIX

Tables and Figures

Tables

Figures

Section 1
Overview

INTRODUCTION

This section describes how GSX-86 works with your microcomputer. The concepts in this section provide background information for the procedures described in later sections.

HOW GSX-86 WORKS

Most graphic devices such as monitors, printers, and plotters draw lines, fill in areas, and produce text differently. GSX-86 manages the differences among these devices and ensures that graphic applications using GSX-86 can communicate with a variety od devices.

Applications written for GSX-86 use the GSX-86 subroutine library, which provides a standard graphic programming interface. GSX-86 also provides device drivers that translate the calls generated by the application to fit the peculiarities of each device. For more details on programming with GSX-86, refer to the *GSX-86 Graphics Extension Programmer's Guide*.

Device Drivers

Because each graphic device is mechanically and electronically different, each requires a special program to communicate with your computer. This program is called a device driver.

GSX-86 gives you a library of device drivers that allows you to use many devices with your microcomputer.

GINSTALL

GINSTALL, the GSX-86 device driver installation program, tells you what device drivers are in the device driver library. You can use GINSTALL to select devices that match the devices on your micorcomputer. After you select devices, GINSTALL creates an assignment file and installs the assignment file and the device drivers on the diskette you specify, which is usually the application diskette.

If you change the devices on your micocomputer, GINSTALL lets you delete and add the appropriate device drivers.

Note: Unless you have a diskette, usually your application diskette, that contains an assignment file and device drivers that match the devices on your computer, you need to use GINSTALL before you use GSX-86. Section 3 contains instructions on how to use GINSTALL.

STARTING YOUR SYSTEM

Set up the monitor, printer, plotter, and any other devices. Refer to your microcomputer and operating system manuals for details on how to install devices and start your system.

Before your use GSX-86, make duplicates of any GSX-86 distribution diskettes. Use the format and copy programs for your operating system. Refer to your operating system manual for instructions on the appropriate procedures and commands.

After you make the duplicates, store the distribution diskettes in a safe place away from heat, magnets, humidity, dust and extreme temperature changes. Use the duplicates as your GSX-86 work diskettes.

You are now ready to turn to Section 2, "Starting GSX-86."

End of Section 1

Section 2
Starting GSX-86

INTRODUCTION

This section explains how you start GSX-86 and delete GSX-86 from memory. A checklist is included in case you have a problem starting GSX-86.

INSTALLING GSX-86

Before using a graphic application that requires GSX-86, an assignment file and device drivers that match your devices must be on your application diskette and GSX-86 must be installed.

Some applications contain an assignment file and a preconfigured set of device drivers that might match the devices on your microcomputer. Refer to the description of the application to learn whether it comes with device drivers or whether you must use GINSTALL to install device drivers. If the application diskette contains device drivers, ensure that they match the devices on your microcomputer. If you have different devices, you must use GINSTALL before you install GSX-86.

Some applications install GSX-86 for you. Refer to the instructions on starting your application to learn whether you must start GSX-86 or whether the application starts it for you.

GSX-86 COMMAND

The command you use to install GSX-86 differs depending on the microcomputer operating system you use. Table 2-1 shows the commands for each operating system GSX-86 supports.

Table 2-1. Commands to Start GSX-86

Operating System	Command
CP/M86 ®	GRAPHICS
Concurrent CP/M ™	GRAPHICS
PC DOS	GSX
MS-DOS	GSX

After you start your system and the operating system prompt is on your monitor, type the command to start GSX-86 in the following form:

‹d1›:COMMAND ‹d2›

The ‹d1› symbol represents the drive identifier of the disk driver where GSX-86 is located. COMMAND represents the command you use to start GSX-86 on your operating system. The ‹d2› symbol represents the drive identifier of the disk drive where the assignment file and the device drivers are located. In most cases, the diskette in ‹d2› is your application diskette. Examples follow.

1. You are using CP/M-86. GSX-86 is on the diskette in drive A, the default drive. Your application diskette, which contains the assignment file and device drivers, is in drive B. In response to the operating system prompt, you type

   GRAPHICS B:

2. You are using PC DOS or MS-DOS. GSX-86 is on the diskette in drive B. Drive A, the default drive, contains your application diskette, assignment file, and device drivers. In response to the operating system prompt, you type

   B:GSX A:

3. You are using Concurrent CP/M. GSX-86 is on the diskette in drive A, the default drive. The diskette in drive A also contains the application, assignment file, and device drivers. In response to the operating system prompt, you type

   GRAPHICS

When you start GSX-86, it is loaded into memory. After GSX-86 is loaded, a copyright message appears on your monitor. You are ready to start your application.

DELETING GSX-86

When you are not using GSX-86, you can free the memory space used by GSX-86. To do this, ensure the operating system prompt is on your monitor. Then type the command to delete GSX-86 from memory. Table 2-2 shows the commands.

Table 2-2. Commands to Delete GSX-86

Operating System	Command
CP/M-86	GRAPHICS N
Concurrent CP/M	GRAPHICS N
PC DOS, version 2.0 and above	GRAPHICS N
MS-DOS, version 2.0 and above	GRAPHICS N

If you are using versions of PC DOS and MS-DOS that precede 2.0, you restart your system to delete GSX-86. To do this, turn the power off on your computer or press and hold the CONTROL, ALTERNATE, and DELETE keys; then release all three keys simultaneously.

Note: Before restarting your system, you should exit the application and save any files you have created.

When you use a command to delete GSX-86 from memory, a copyright notice appears on your monitor followed by the message

GSX-86 is not installed

You can also receive the above message if you cannot install GSX-86.

ERROR MESSAGES

When you start GSX-86, you can receive error messages. The format of the messages differs according to your operating system. CP/M-86 echoes what you type followed by a question mark, ? Concurrent CP/M echoes what you type followed by a question mark, ?, and a brief message. PC DOS and MS-DOS tell you that you entered a bad command or filename.

If you receive any of the above error messages when you start GSX-86, one of the following conditions exists:

- You typed the command incorrectly.
- GSX-86 is not on the diskette in the drive you identified in the command.

Either retype the command or insert the diskette that contains GSX-86 in the correct drive.

Refer to Appendix A for a description of error messages and steps you can take to correct the errors.

INSTALLATION CHECKLIST

If you have a problem, starting GSX-86, use the checklist below to ensure you have completed all necessary steps.

- Are the display monitor, printer, plotter and other devices properly attached to your computer?
- Is the operating system prompt on your monitor?
- Is the diskette containing GSX-86 in the disk drive you specified in the command to start GSX-86?
- Did you specify the correct drive identifier of the drive where the assignment file and device drivers are located?
- Did you type the correct command to start GSX-86?

End of Section 2

Section 3
Introduction to GINSTALL

INTRODUCTION

GINSTALL, the GSX-86 device driver installation program, lets you install device drivers for a variety of display monitors, printers, plotters, and specially designed computer output cameras for use with graphic applications that use GSX-86.

GINSTALL provides menus from which you can select drivers for the devices you are using. GINSTALL uses your menu selections to create or update the assignment file, described below.

ASSIGNMENT FILE

The assignment file, named ASSIGN.SYS, lists the device drivers of the devices you select, and communicates this information to GSX-86.

GINSTALL creates or updates the assignment file on the application diskette or, if the application diskette does not have sufficient space for these files, on another diskette.

SYSTEM REQUIREMENTS

To use GINSTALL, your system must contain either:

- two floppy diskette drives
- a hard disk drive and one floppy diskette drive

USING GINSTALL MENUS

GINSTALL contains menus that let you do the following:

- select a device for addition to the assignment file
- select a device for deletion from the assignment file
- display the current device selections

- update the assignment file
- exit GINSTALL and return to the operating system

Selecting Options

GINSTALL displays two types of menus: numbered lists and queries.

To select an option from a numbered list, type its number after the prompt

Enter Option Number -

and press the RETURN key.

The query menus require a Yes or No response. Type Y or N, and press the RETURN key

Correcting Typing Errors

If you mistype a response, use the BACKSPACE key to move the cursor to its original position. The BACKSPACE key does not erase characters. Type the correct response over the error, and press the RETURN key.

Returning to the MAIN MENU

All of the numbered list menus that follow the MAIN MENU let you press the ESCAPE (ESC) key to return to the MAIN MENU. When you do, GINSTALL cancels any selection in process.

GINSTALL on the Default Drive

Before you start GINSTALL, insert the diskette containing GINSTALL in the default drive. If you have copied GINSTALL to a hard disk, set your default drive to the hard disk.

STARTING GINSTALL

To start GINSTALL, type the following command in response to the operating system prompt:

GINSTALL

When GINSTALL starts, a copyright message, followed by a brief description of GINSTALL, appears on your monitor. Press the RETURN key to display the first menu.

SPECIFY DISK DRIVES MENU

GINSTALL'S first menu, SPECIFY DISK DRIVES, in Figure 3-1, asks you to identify the drives containing the application diskette and the device drive diskette. If the application diskette does not have sufficient space for the assignment file and device driver files, you can use another diskette in place of the application diskette.

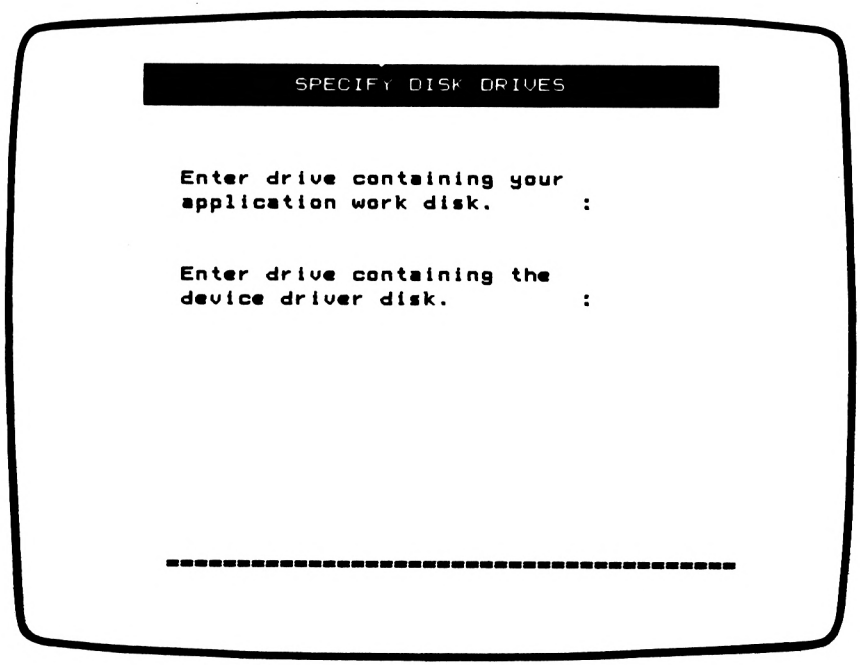


Figure 3-1. SPECIFY DISK DRIVES

Type the drive identifier for the drive containing the diskette on which GINSTALL will create or update the assignment file and device driver files, and press the RETURN key. The drive identifier can be a letter from A to P.

Type the drive identifier for the drive containing the device driver diskette, and press the RETURN key. After a pause the MAIN MENU appears. The MAIN MENU and its functions are described in Section 4.

End of Section 3

Section 4
GINSTALL Functions

INTRODUCTION

This section describes the functions contained in GINSTALL'S MAIN MENU. The functions are described in order in which they appear on the menu.

MAIN MENU FUNCTIONS

The MAIN MENU functions let you do the following:

- add devices to the assignment file
- delete devices from the assignment file
- display the current selections in the assignment file
- update the assignment file and copy device driver files to the application diskette
- exit GINSTALL and return to the operating system

The MAIN MENU is illustrated in Figure 4-1.

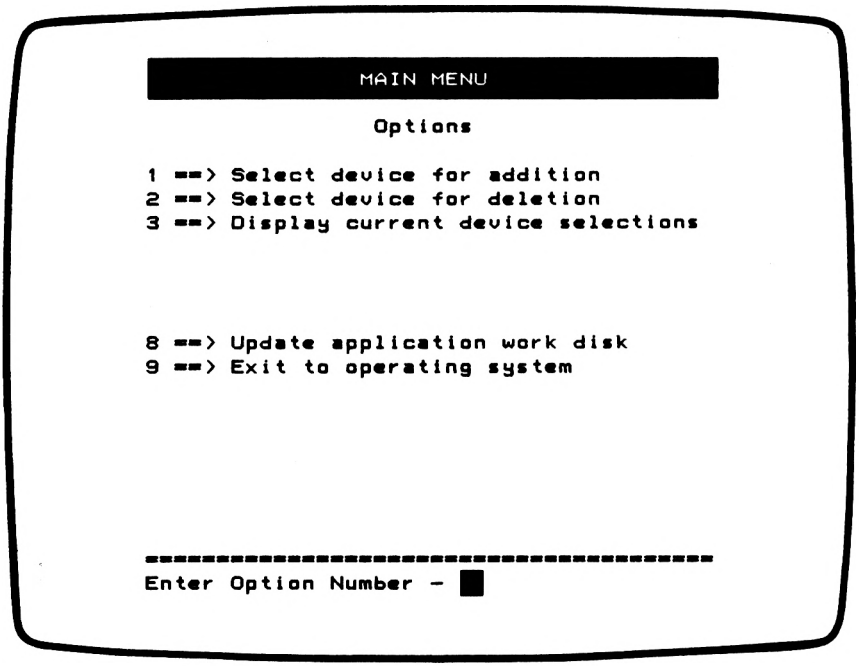


Figure 4-1. MAIN MENU

To select a function from the MAIN MENU, type its option number after the Enter Option Number prompt, and press the RETURN key

ADDITIONAL MENUS

When you select the SELECT DEVICE FOR ADDITION or the SELECT DEVICE FOR DELETION function, GINSTALL displays additional menus. These menus are named and described in the descriptions of the two SELECT functions.

SELECTING DEVICE FOR ADDITION

The SELECT DEVICE FOR ADDITION function lets you add devices to the assignment file. However, GINSTALL does not change the assignment file or copy device driver files until you return to the MAIN MENU and choose the UPDATE APPLICATION WORK DISK function.

SELECT DEVICE CATEGORY Menu

The first menu you see when you select option number 1 from the MAIN MENU is the SELECT DEVICE CATEGORY FOR ADDITION Menu, illustrated in Figure 4-2.

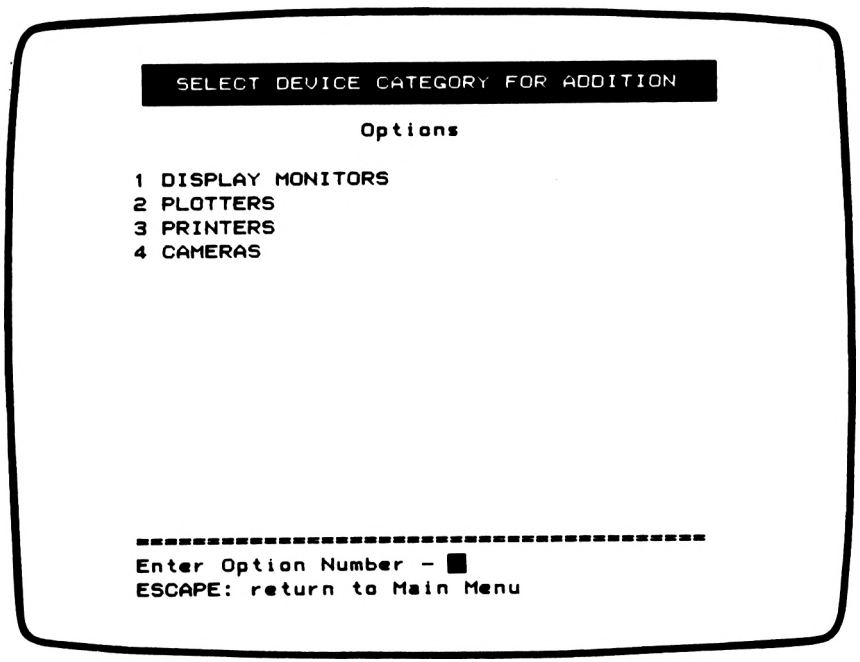


Figure 4-2. SELECT DEVICE CATEGORY FOR ADDITION

To select one of the device categories type its option number. Press the RETURN key. GINSTALL then displays the device menu for the category you selected.

Error Message

You can receive the following error message when selecting a device category. GINSTALL displays the message at the bottom of the SELECT DEVICE CATEGORY FOR ADDITION Menu.

Selected category is full. You must delete a device from the category before adding another.
Press RETURN when ready to proceed.

The assignment file has a limit to the number of devices in each device category that it can contain. If you try to select a category that contains its limit, GINSTALL displays the above message.

Press the RETURN key to return to the MAIN MENU. To delete a device, use the SELECT DEVICE FOR DELETION function, MAIN MENU selection number 2.

Device Menus

The device menus for display monitors, plotters, printers, and cameras list the devices for which driver files exist. An option number precedes a brief description of each device. Figures 4-3 through 4-6 are examples of typical device menus.

To select a device from the menu, type its option number after the Enter Option Number prompt, and press the RETURN key.

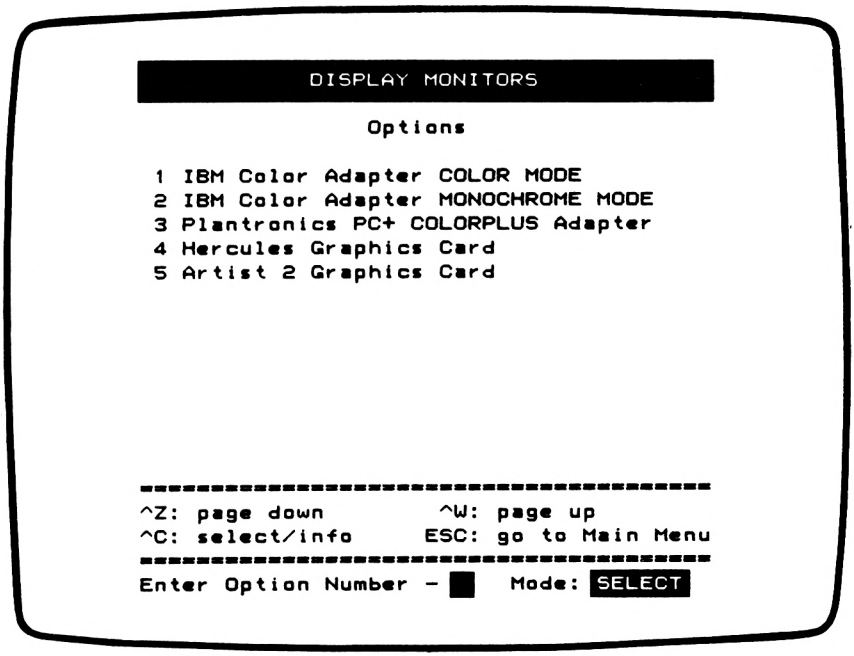


Figure 4-3. DISPLAY MONITORS

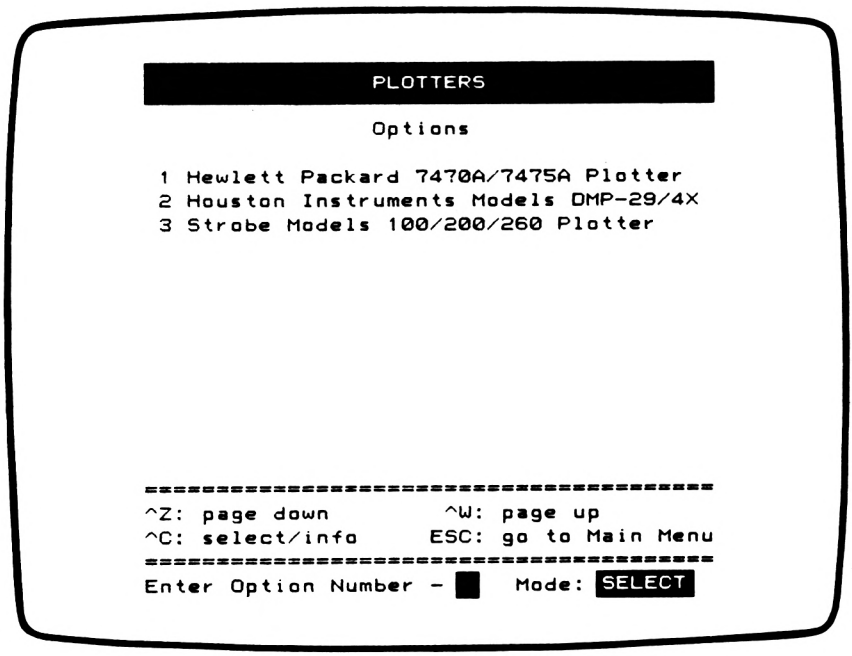


Figure 4-4. PLOTTERS

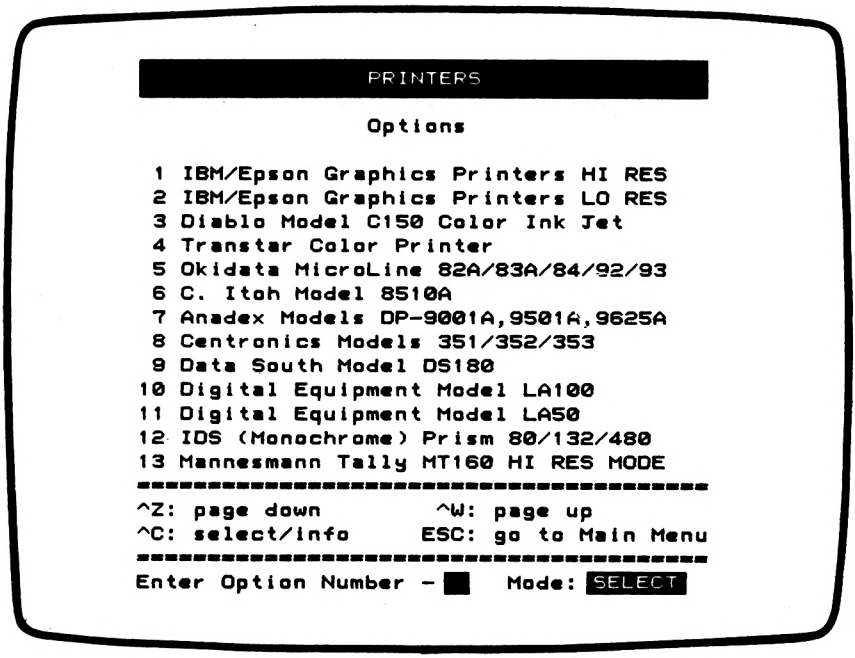


Figure 4-5a. PRINTERS Page 1

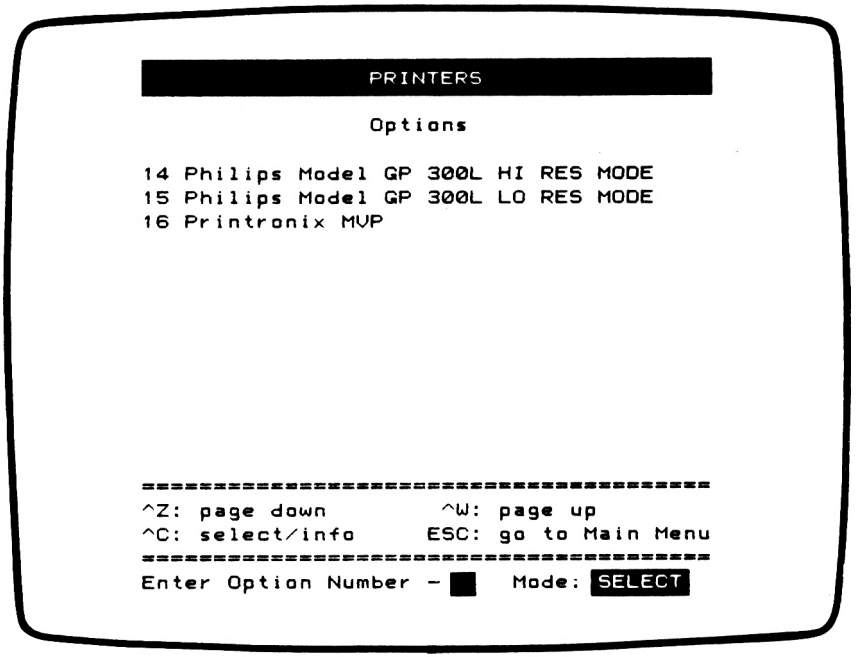


Figure 4-5b. PRINTERS Page 2

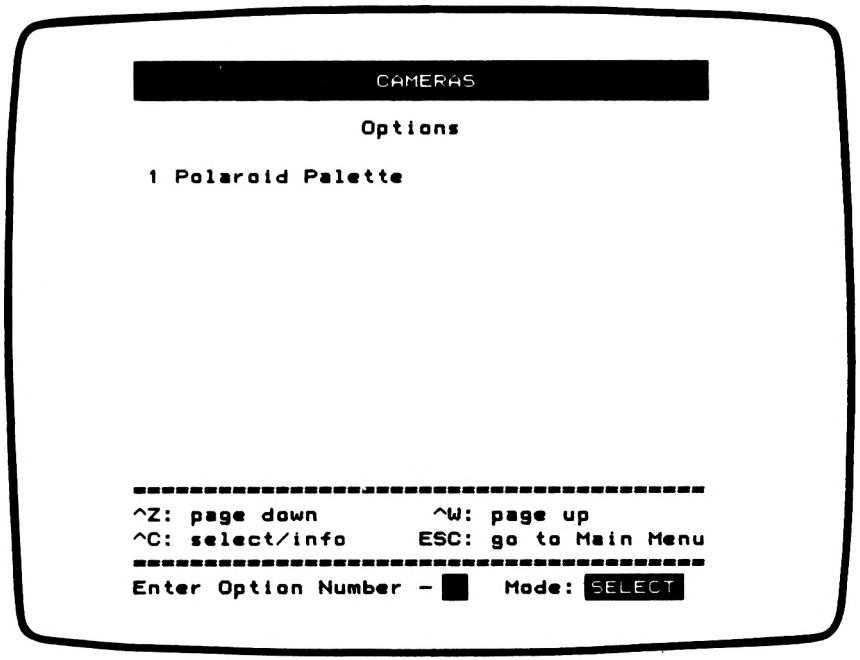


Figure 4-6. CAMERAS

Special Keystrokes

At the bottom of the device menu screen you see a small informational panel describing several keystrokes you can use. Table 4-1 names the keystrokes and describes their actions.

TABLE 4-1. DEVICE MENU KEYSTROKES

KEYSTROKE	ACTION
^Z	Displays next page of menu. For example, the printer menu (Figure 4-5a and 4-5b) is two pages long. If no next page exists, ^Z has no effect.
^W	Displays previous page of menu. If no previous page exists, ^W has no effect.
^C	Switches between SELECT and INFO modes described below.
Esc	Cancels current selection and returns to MAIN MENU.

SELECT AND INFO Modes

For each device menu, GINSTALL supports two modes, SELECT and INFO. A reverse video rectangle at the bottom of the menu tells you the current mode.

When the rectangle says SELECT, you can select a device by typing its option number and then pressing the RETURN key.

To change from SELECT to INFO mode, press ^C. In INFO mode, when you type an option number and press the RETURN key, GINSTALL displays information describing the device. Figure 4-7 shows an example.

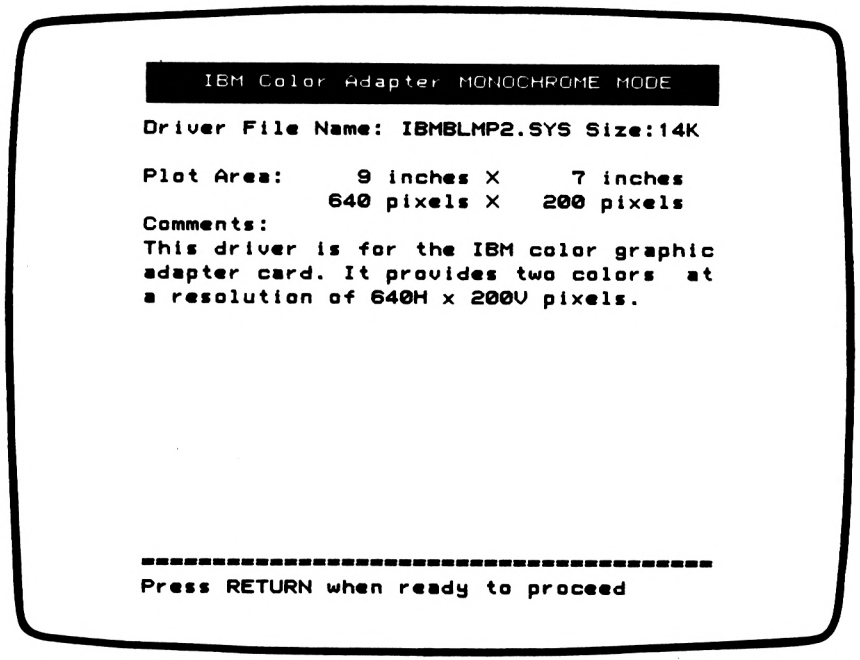


Figure 4-7. IBM Color Adapter MONOCHROME MODE

To return to the device menu, press the RETURN key. You remain in INFO mode.

You cannot select a device while you are in INFO mode. To return to SELECT mode, press C

Subsequent menus

The type of device you select determines which menus GINSTALL displays next. For example, if you select a display monitor, you see one or more of the following menus:

- SELECT MOUSE OPTION FOR DISPLAY MONITOR
- MOUSE COMMUNICATION PORT
- SET PRIMARY DEVICE

If you select a plotter, printer or camera you see one or more of the following menus:

- PLOTTER, PRINTER, or CAMERA COMMUNICATION PORT
- SET PRIMARY DEVICE

Each of these menus is described in this section

SELECT MOUSE OPTION Menu

When you select a display monitor, GINSTALL displays the SELECT MOUSE OPTION FOR DISPLAY MONITOR Menu, in Figure 4-8. In addition to listing several mice, the menu gives you the option of not using a mouse.

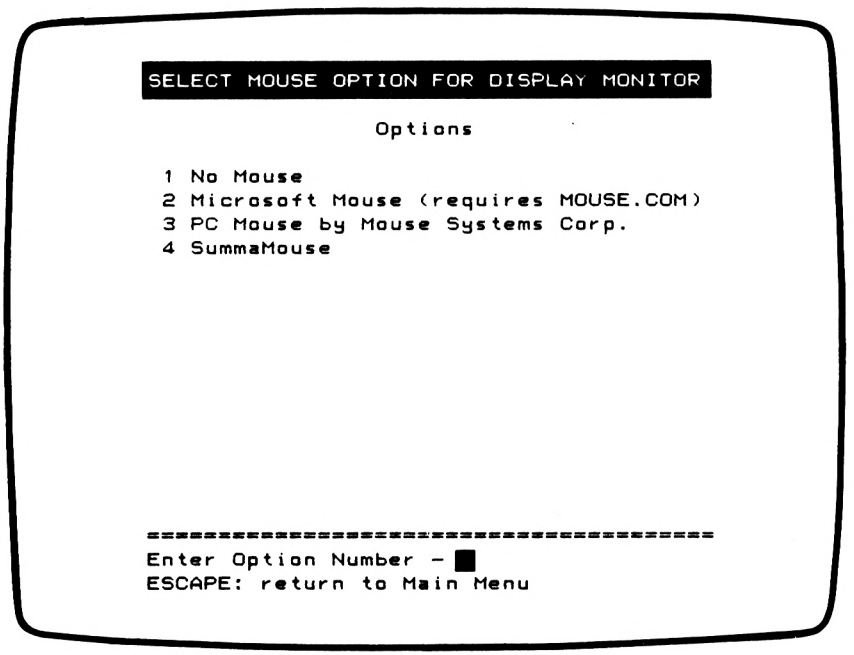


Figure 4-8. SELECT MOUSE OPTION FOR DISPLAY MONITOR

If you select the Microsoft Mouse, you must copy the file MOUSE.COM to your application diskette. MOUSE.COM is supplied on a diskette that comes with the Microsoft Mouse.If you select a mouse that comes with its own interface board and communication port (such as the Microsoft Mouse) or the No Mouse option, GINSTALL does one of the following:

- Displays the SET PRIMARY DEVICE Menu, if the assignment file already lists a display monitor. The SET PRIMARY DEVICE menu is described later in this section.
- Completes the selection and returns you to the MAIN MENU, if the assignment file does not already list a display monitor.

If you select one of the other mouse options, GINSTALL displays the MOUSE COMMUNICATION PORT Menu, below.

MOUSE COMMUNICATION PORT Menu

With this menu you identify the communication port to which the mouse is connected. Without this information, GSX-86 does not know where to look for mouse input. Figure 4-9 shows a MOUSE COMMUNICATION PORT menu.

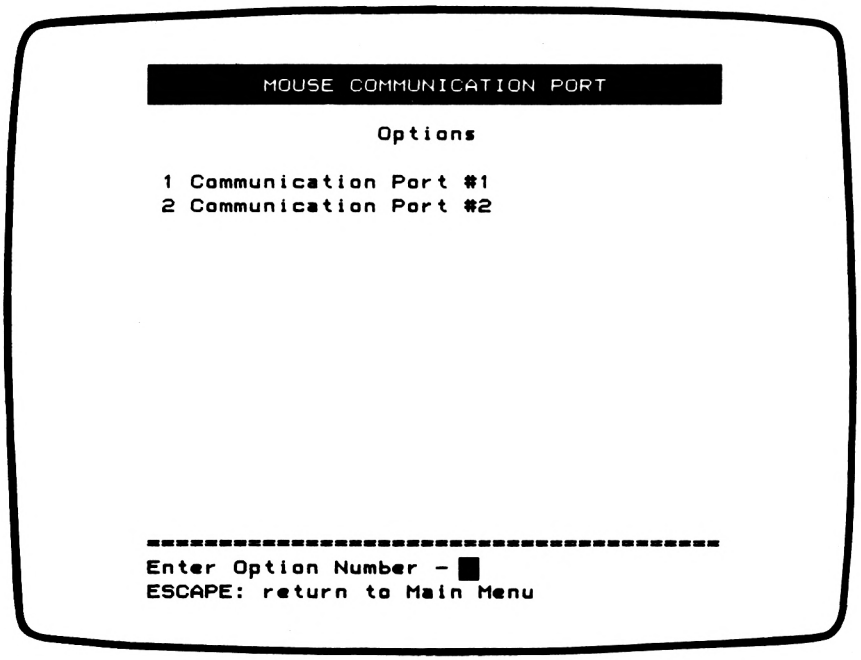


Figure 4-9. MOUSE COMMUNICATION PORT

After you select the communication port, GINSTALL either:

- Displays the SET PRIMARY DEVICE Menu, if the assignment file already lists a display monitor. The SET PRIMARY DEVICE menu is described later in this section.

- Completes the selection and returns you to the MAIN MENU, if the assignment file does not already list a display monitor.

PLOTTER, PRINTER, and CAMERA COMMUNICATION PORT Menus

If you select a plotter, printer or camera, GINSTALL displays a communication port menu immediately after you select the device. The menu identifies the port to which the plotter, printer, or camera is connected. Without this information, GSX-86 does not know where to send plotter, printer, or camera output.

Type the communication port's option number, and press the RETURN key.

If the assignment file already lists a plotter, printer, or camera, GINSTALL displays the SET PRIMARY DEVICE menu.

However, if the assignment file does not list another plotter, printer, or camera, the selection is complete. GINSTALL returns you to the MAIN MENU.

SET PRIMARY DEVICE Menu

The SET PRIMARY DEVICE Menu lets you name the newly-selected device as the primary device for its category. The primary device is the device that an application uses when the assignment file lists more than one device of any type.

For example, if your system has two display monitors, the application directs all output to the primary display unless:

- The application permits you to direct output to a secondary device
- You explicitly direct the output to the secondary device.

Changing the Primary Device

The SET PRIMARY DEVICE Menu, in Figure 4-10, identifies the device category, the primary device, and the device you just selected. The prompt asks if you want the new device to become the primary device. Type Y (Yes) or N (No), and press the RETURN key.

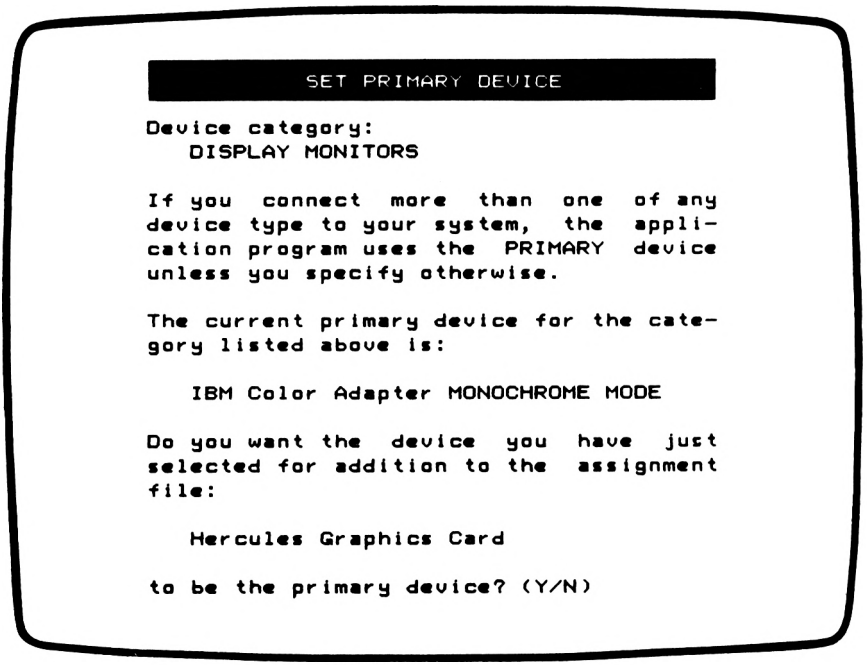


Figure 4-10. SET PRIMARY DEVICE

If you type Y, the new device becomes the primary device.

If you type N, the current primary device remains the primary device.

After you respond to the SET PRIMARY DEVICE menu, the selection is complete. GINSTALL returns you to the MAIN MENU.

SELECTING DEVICE FOR DELETION

The SELECT DEVICE FOR DELETION function lets you delete a device from the assignment file. However, GINSTALL does not change the assignment file or delete device driver files until you return to the MAIN MENU and choose the UPDATE APPLICATION WORK DISK function.

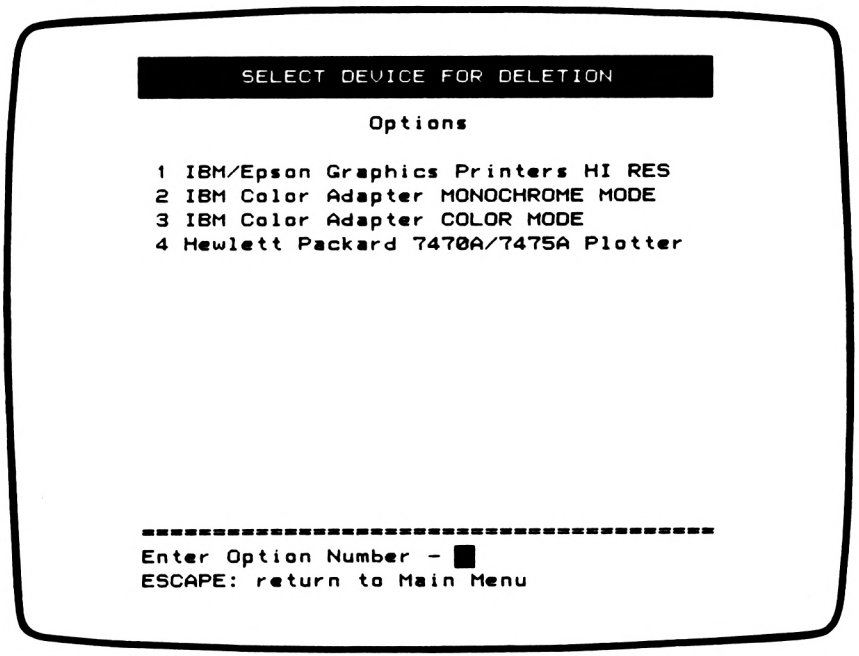


Figure 4-11. SELECT DEVICE FOR DELETION

Type the option number of the device you want to delete, and press the RETURN key. GINSTALL selects the device for deletion and returns you to the MAIN MENU.

DISPLAY DEVICE SELECTIONS

The DISPLAY CURRENT DEVICE SELECTIONS function displays a list of the device drivers currently selected for the assignment file. Figure 4-12 shows a typical display.

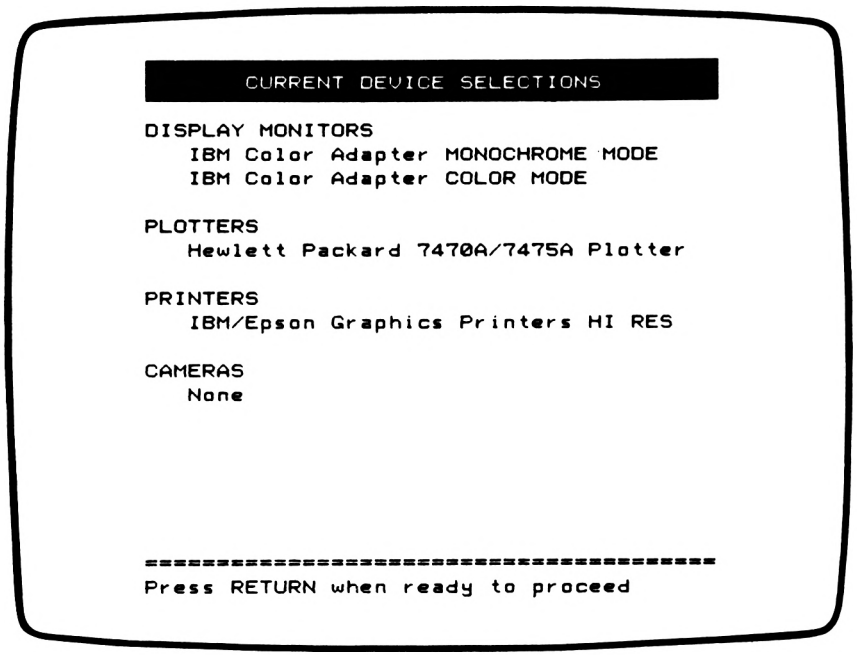


Figure 4-12. Initial CURRENT DEVICE SELECTIONS

To return to the MAIN MENU, press the RETURN key.

If you select the DISPLAY function either

- before you add or delete any devices or
- immediately after you update the assignment file

you see the assignment file list as it currently exists on your application diskette.

However, if you select the DISPLAY function after adding or deleting devices, you see the assignment file list as it would appear if you updated the assignment file at that moment.

For example, suppose Figure 4-12 shows the devices currently listed in your assignment file. If you delete the IBM®Color Adapter COLOR MODE display monitor, you see the display shown in Figure 4-13 the next time you select the DISPLAY function.

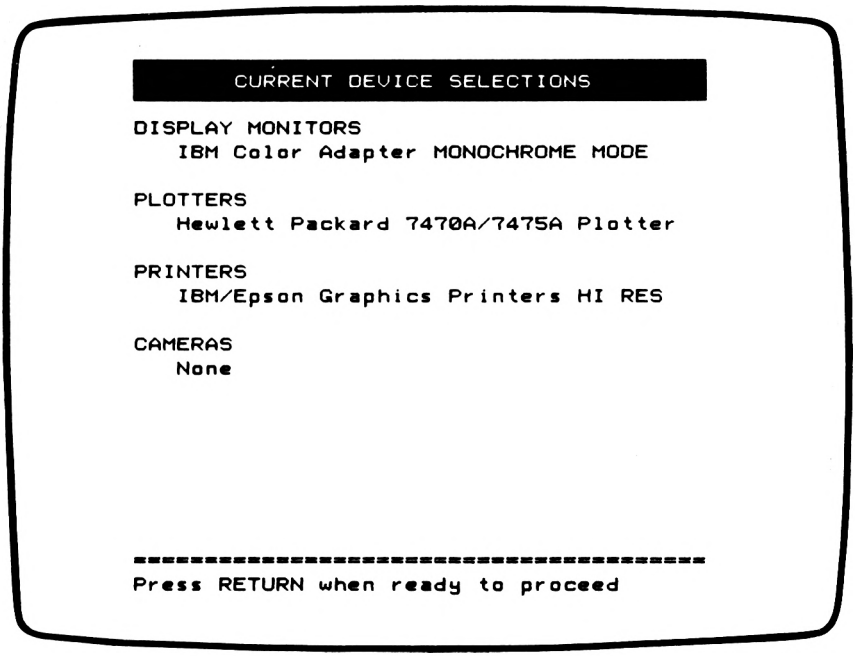


Figure 4-13. New CURRENT DEVICE SELECTIONS

Although the Color Adapter COLOR MODE monitor no longer appears in the list GINSTALL does not delete the device until you select the UPDATE function.

UPDATING YOUR SELECTIONS

The UPDATE/APPLICATION WORK DISK function deletes and/or adds the devices you selected./

When you select the UPDATE function, GINSTALL follows this sequence:

1. Deletes any device driver files you selected. The name of the device currently being deleted flashes on and off.
2. Adds any device driver files you selected. The name of the device currently being added flashes on and off.
3. Rewrites the assignment file.

Figures 4-14a and 4-14b show you the display you see on the monitor at two points during an update.

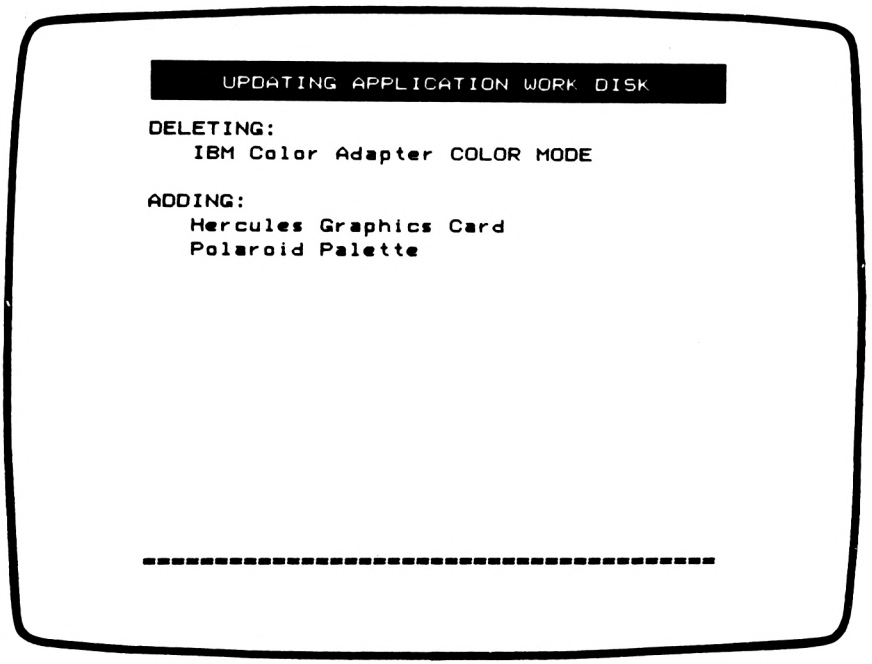


Figure 4-14a. Deleting and Adding Device Driver Files

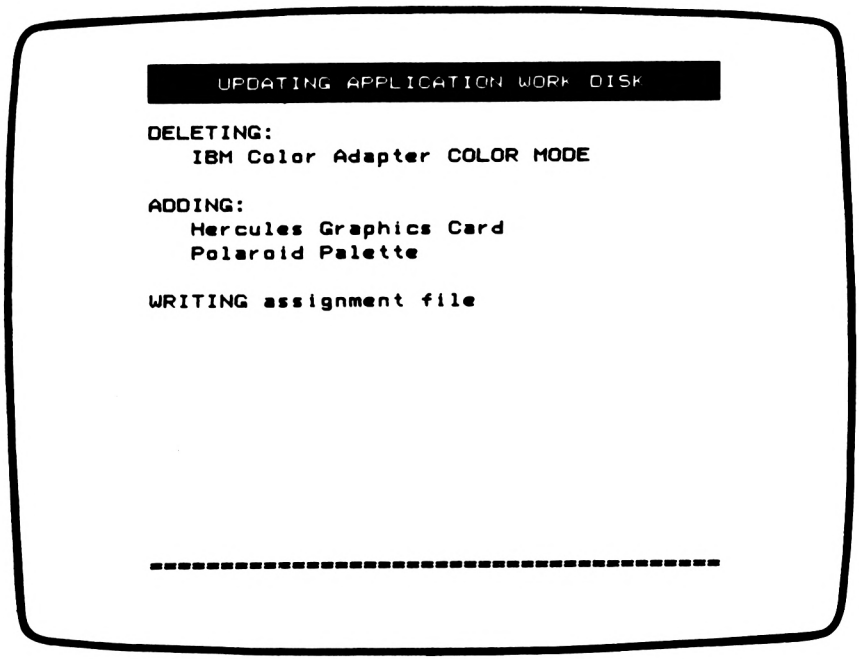


Figure 4-14b. Writing the Assignment File

After deleting or adding device drivers and rewriting the assignment file, GINSTALL displays the message

Press RETURN when ready to proceed

When you press the RETURN key, GINSTALL returns you to the MAIN MENU.

Changing Device Driver Diskettes

During an update, GINSTALL copies driver files from a device driver diskette to a diskette you specified when you started GINSTALL. If the needed driver file is not on the current device driver diskette, GINSTALL displays the following message:

Current selection not on this device driver library disk. Please insert other device driver library disk.
Press RETURN when ready to proceed.

Remove the driver library diskette, insert the library diskette that contains the driver file, and press the RETURN key.

If you do not change diskettes, or if the diskette you insert does not contain the needed driver file, GINSTALL displays the following message when you press the RETURN key.

ERROR; Current selection not found.
Device driver file: D:NNNNNNNN.SYS
This selection abandoned.
Press RETURN when ready to proceed.

D:NNNNNNNN.SYS is the driver identifier and filename for the driver file of the device you selected.

GINSTALL abandons the current selection. When you press the RETURN key, GINSTALL continues the updates with the next driver file or writes the new assignment file.

GINSTALL Not In Default Drive

If the diskette containing GINSTALL is not in the default drive when the update is complete, GINSTALL displays the message:

Update complete. Please insert other device driver library disk.
Press RETURN when ready to proceed.

Insert the diskette containing GINSTALL, and press the RETURN key. The message disappears and the RETURN key prompt moves below the dashed line. Press the RETURN key again to return to the MAIN MENU.

Warning Messages

When you select the UPDATE APPLICATION WORK DISK function, GINSTALL checks your selections for the number of devices of each type. If the updated assignment file contains:

- no display monitor
- more than two display monitors

- more than one plotter, printer, or camera

GINSTALL then displays the warning messages in Figures 4-15 and 4-16 before updating the application diskette.

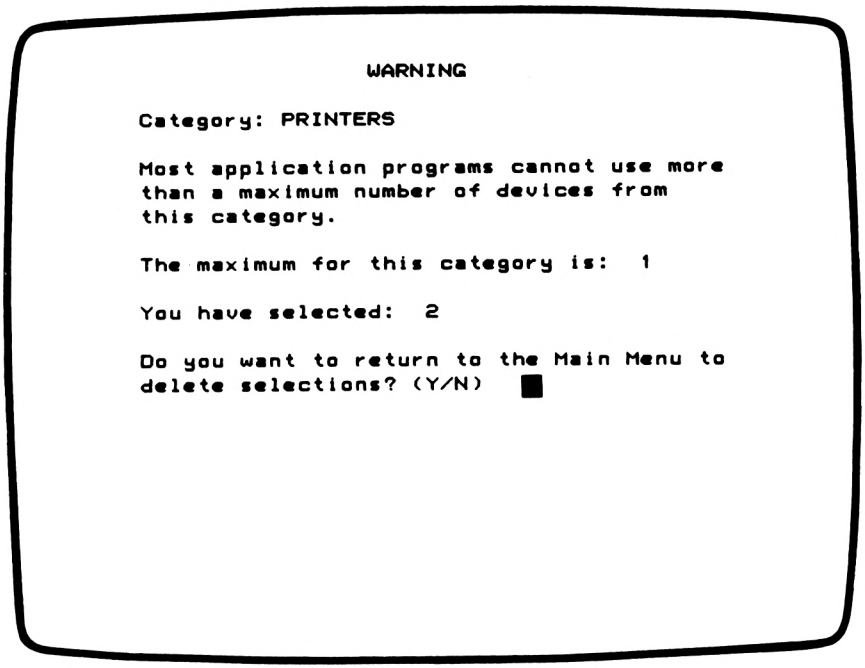


Figure 4-15. Maximum Device Number Warning

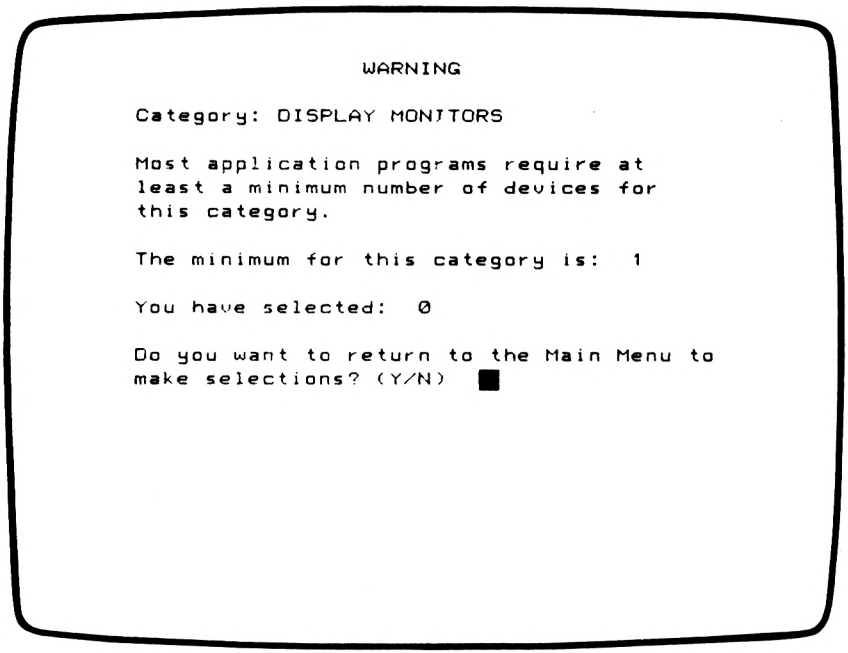


Figure 4-16. Minimum Device Number Warning

The warnings give you the following information:

- the device category
- the maximum or minimum number of devices for the category
- the number of devices you selected

The prompt at the end of the warning asks if you want to return to the MAIN MENU to add or delete selections. Type Y (Yes) or N(No), and press the RETURN key.

If you type Y, GINSTALL cancels the update request and returns you to the MAIN MENU. To add a device, select option number 1 from the MAIN MENU. To delete a device, select option number 2 from the MAIN MENU.

If you type N (No), GINSTALL updates the assignment file according to your selections.

If you delete all you display monitors or if you have less than the minimum number of devices requires in another category, you might encounter one of the following:

- you cannot install GSX-86
- you can install GSX-86 but cannot start your application. Instead, you receive the following message:

No graphics input, press RETURN

If either situation occurs, you must return to GINSTALL and add the missing device drivers.

EXIT TO OPERATING SYSTEM

The EXIST TO OPERATING SYSTEM function exits GINSTALL and returns you to the operating system prompt on the current default drive.

Error Message

If you select the EXIT function after you add or delete devices, but before you update the assignment file, GINSTALL displays the message:

DEVICES SELECTED TO ADD OR DELETE
Abandon selections? (Y/N) N

GINSTALL prompts an N for No. Press the RETURN key, to return to the MAIN MENU. You can then select option number 8 and update the assignment file.

If you type Y for Yes over the N prompt, GINSTALL abandons the pending selections and returns you to the operating system.

End of Section 4

Appendix A
GSX-86 Error Messages

ERROR MESSAGES

You can receive an error message when you use GSX-86 for one of the following reasons:

- The wrong diskette is in the disk drive specified in the error message.
- The file specified in the error message is damaged and needs to be regenerated.
- Not enough memory for GSX-86 and device drivers.

The error messages and solutions are discussed below.

---

d:ASSIGN.SYS not found
d:ffffffff.SYS not found
d:ffffffff.SYS close error

All of these error messages tell you that GSX-86 cannot find a file it needs. Generally, the file cannot be found because the diskette in the drive specified does not contain the file specified in the message.

The d represents the disk drive identifier. ASSIGN.SYS is the name of the assignment file GINSTALL creates. The ffffffff.SYS portion of a message represents the filename of a missing file.

To correct the error, insert the diskette that contains the ASSIGN.SYS file and the device driver files and proceed.

---

d:ASSIGN.SYS syntax error
d:ffffffff.SYS empty
d:ffffffff.SYS contains absolute segment

All of these error messages tell you the file specified in the message is empty or damaged.

The d represents the disk drive identifier. ASSIGN.SYS is the name of the assignment file GINSTALL creates. The ffffffff.SYS portion of a message represents the filename of the empty or damaged file.

To restore the file, use GINSTALL to update the diskette containing the file. Refer to the instructions on using GINSTALL in Section 4 of this guide. After you use GINSTALL, start GSX-86 and your application again.

Not enough memory for GSX-86

This error message tells you your computer does not have enough Random Access Memory.

To use GSX-86 and your applications, you need to add more Random Access Memory. The amount of memory required varies with the computer operating system, and applications you are using. Consult your computer dealer for more details.

End of Appendix A

GSX™
Graphics Extension
Programmer's Guide

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

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

Foreword

MANUAL OBJECTIVE

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

INTENDED AUDIENCE

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

MANUAL DESIGN

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

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

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

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

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

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

Appendixes contain the following reference information:

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

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

Appendix C - The Virtual Device Interface (VDI) specification

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

Table of Contents

Table of Contents (continued)

Appendixes

Appendixes (continued)

Appendixes (continued)

Appendixes (continued)

Tables and Figures

Section 1
INTRODUCTION

ABOUT THIS MANUAL

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

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

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

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

GSX BENEFITS

GSX adds graphics to your operating system, as follows:

- GSX supports DR Graph..and DR Draw., two products that extend your graphics capability. DR Graph allows you to graph and plot data by making simple menu selections. DR Draw lets you draw complex graphics images.

- GSX opens a world of application software. You can run any graphics application program that uses GSX with several 8080 and 8086 microcomputer operating systems.

- GSX promotes user portability. The interface between you and GSX is identical to the interface between you and your operating system.

- GSX provides a device-independent software interface for your application programs. You will not need to rewrite your programs if you decide to use a printer instead of a plotter, for example.

GSX FUNCTIONS

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

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

Transforming Points

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

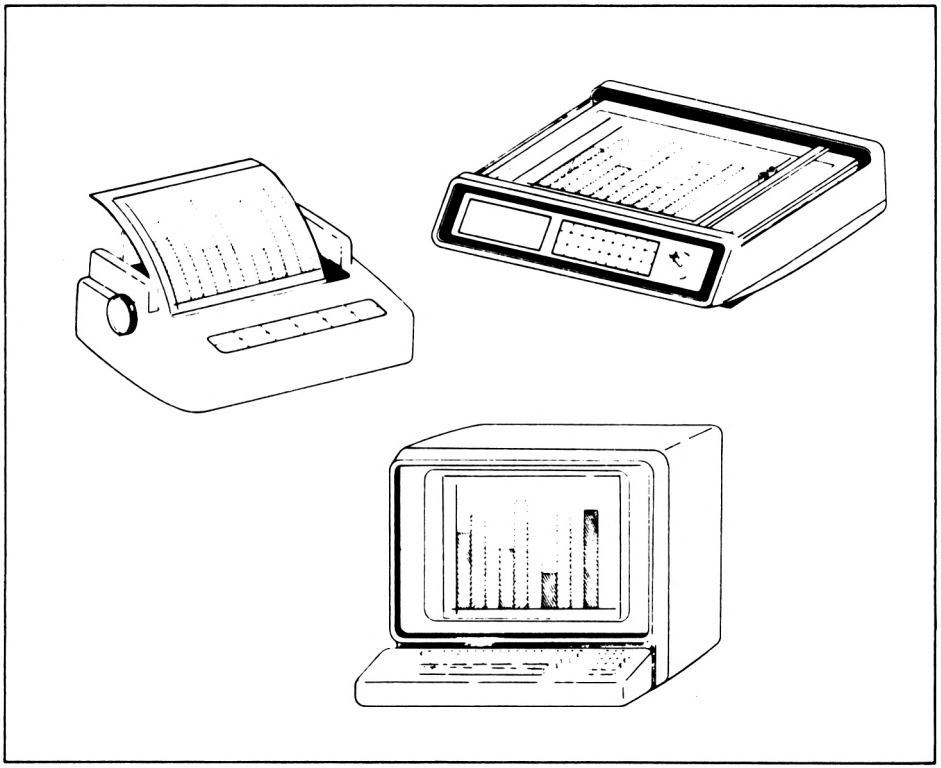


Figure 1-1. GSX Provides Device-Independent Graphics

Servicing Graphics Requests

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

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

Loading Device Drivers

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

End of Section 1

Section 2
PROGRAMMER'S OVERVIEW

INTRODUCTION

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

GRAPHICS SYSTEM EXTENSION ARCHITECTURE

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

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

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

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

Graphics Device Operating Systems (GDOS)

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

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

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

Graphics Input/Output System (GIOS)

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

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

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

Enabling Graphics

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

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

GRAPHICS MODE INITIALIZATION

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

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

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

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

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

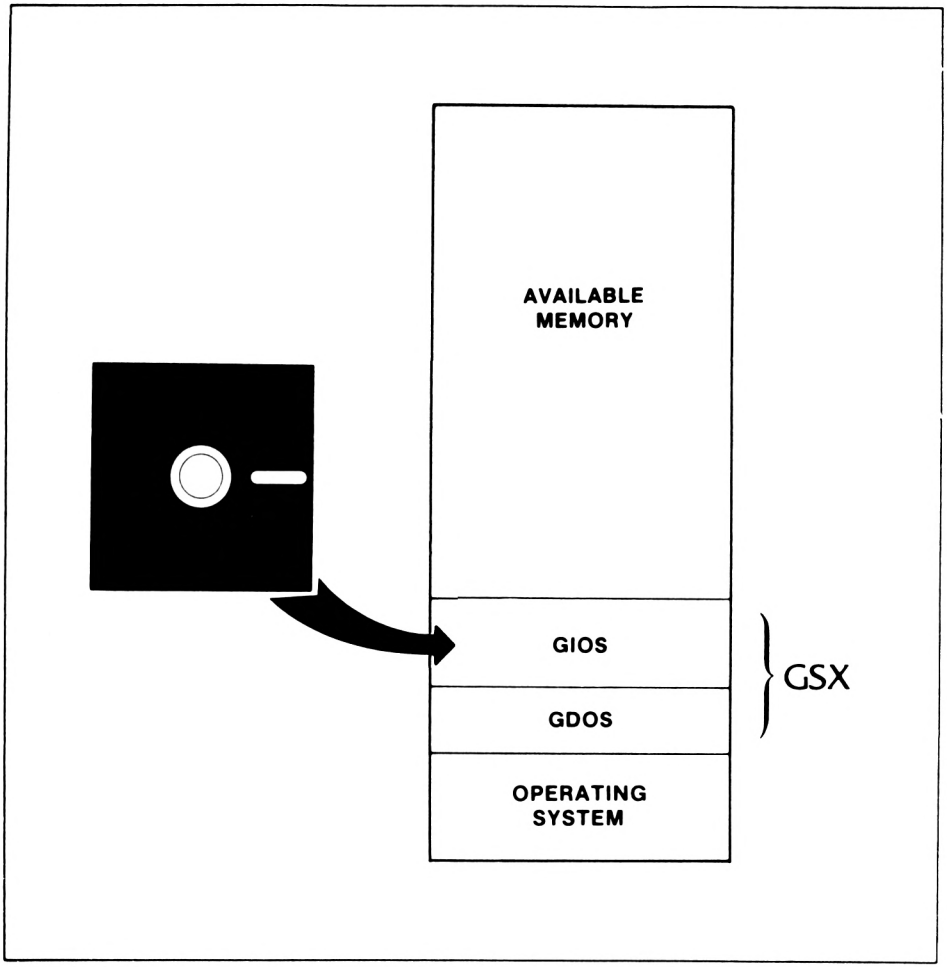


Figure 2.1 GSX Memory Map

APPLICATION PROGRAMS

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

End of Section 2

Section 3
GDOS

INTRODUCTION

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

GDOS FUNCTIONS

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

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

Graphics Calls

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

Dynamic Loading

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

Transforming Points

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

GDOS CALLING SEQUENCE

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

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

GDOS OPCODES

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

Table 3-1. GSX Operation Codes

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

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

Table 3-1. (continued)

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

Table 3-1. (continued)

Opcode	Description
11	GENERALIZED DRAWING PRIMITIVE displays a generalized drawing primitive.

ID	Definition
1	BAR
2	ARC
3	PIE SLICE
4	CIRCLE
5	PRINT GRAPHIC CHARACTERS
6-7	RESERVED (for future use)
8-10	UNUSED (and available)

Opcode	Description
12	SET CHARACTER HEIGHT sets text size.
13	SET CHARACTER UP VECTOR sets text direction.
14	SET COLOR REPRESENTATION defines the color associated with a color index.
15	SET POLYLINE LINETYPE sets linestyle for polylines.
16	SET POLYLINE LINEWIDTH sets width of lines.
17	SET POLYLINE COLOR INDEX sets color for polylines.
18	SET POLYMARKER TYPE sets marker type for polymarkers.
19	SET POLYMARKER SCALE sets size for polymarkers.
20	SET POLYMARKER COLOR INDEX sets color for polymarkers.
21	SET TEXT FONT sets device-dependent text style.
22	SET TEXT COLOR INDEX sets color of text.

Table 3-1. (continued)

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

LOADING GIOS FILES

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

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

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

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

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

Assignment Table Format

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

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

DDXd:filename;comments

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

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

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

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

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

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

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

Memory Management

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

End of Section 3

Section 4
GIOS

INTRODUCTION

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

PURPOSE OF GIOS

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

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

GIOS FUNCTIONS

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

VIRTUAL DEVICE INTERFACE SPECIFICATION

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

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

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

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

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

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

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

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

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

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

CREATING A GIOS FILE

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

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

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

End of Section 4

Section 5
OPERATING PROCEDURES

INTRODUCTION

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

GSX DISTRIBUTION FILES

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

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

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

RUNNING GRAPHICS APPLICATIONS UNDER GSX

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

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

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

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

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

DETERMINING MEMORY REQUIREMENTS

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

GSX stack requirements:

Open workstation call = approximately 500 bytes

All others = Ptsin size + 128

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

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

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

DEBUGGING GRAPHICS APPLICATIONS UNDER GSX

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

WRITING A NEW DEVICE DRIVER

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

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

End of Section 5

Appendix A
GSX CALLING CONVENTIONS FOR CP/M

INTRODUCTION

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

GSX SKELETON DEVICE DRIVER

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

FORMAT

Function: GSX skeleton device driver

Input Parameters

contrl(1) -- Opcode for driver function

contrl(2) -- Number of vertices in array ptsin. Each vertex consists of an x and a y coordinate so the length of this array is twice as long as the number of vertices specified.

contrl(4) -- Length of integer array intin

contrl(6-n) -- Opcode dependent information

intin -- Array of integer input parameters

ptsin -- Array of input coordinate data

Output Parameters

contrl(3) -- Number of vertices in array ptsout. Each vertex consists of an x and a y coordinate so the length of this array is twice as long as the number of vertices specified.

contrl(5) -- Length of integer array intout

contrl(6-n) -- Opcode dependent information

intout -- Array of integer output parameters

ptsout -- Array of output coordinate data

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

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

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

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

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

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

Parameter Block (address is passed in DE):

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

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

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

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

GDOS CALLING CONVENTIONS

The GDOS calling sequence is summarized below.

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

Parameter Block Contents:

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

Control Array on Input:

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

Input Parameter Array:

intin -- Array of input parameters

Input Coordinate Array:

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

End of Appendix A

Appendix B
GSX CALLING CONVENTIONS FOR CP/M-86, IBM PC DOS, AND MS-DOS

INTRODUCTION

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

GDOS CALLING SEQUENCE

The GDOS calling sequence is outlined below.

Access via interrupt 224

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

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

Parameter Block Contents:

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

Control Array on Input:

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

Input Parameter Array:

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

Input Point Coordinate Array:

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

Control Array on Output:

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

Output Parameter Array:

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

Output Point Coordinate Array:

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

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

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

INVOKING DEVICE DRIVERS

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

```
CGroup          Group   Driver_Code

Driver_Code     CSeg
                Public Driver

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

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

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

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

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

        Callf   Dd_Driver           ; Invoke the driver with Ds:Dx

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

StackBase               Dw          Seg Top_Stack

Dd_Driver_Code  CSeg
                Extrn   Dd_Driver   :Far

Stack           SSeg
                Rs      16          ; This module pushes 8 words

; Top_Stack is defined in the last module linked in.

Extrn   Top_Stack       :Byte

                End
```

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

```
A>REN GIOSXX.SYS=GIOSXX.CMD
```

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

ERROR MESSAGES

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

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

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

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

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

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

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

GSX CS:IP GIOS invalid

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

GSX CS:IP Illegal function: (Cx)

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

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

End of Appendix B

Appendix C
VIRTUAL DEVICE INTERFACE (VDI) SPECIFICATION

INTRODUCTION

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

FORMAT

Function: GSX graphics operation

Input Parameters

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

intin -- Array of integer input parameters.
ptsin -- Array of input point coordinate data.

Output Parameters

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

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

Notes

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

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

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

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

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

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

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

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

OPEN WORKSTATION	Initialize a graphic workstation.	
Input	contrl(1) --	Opcode = 1
	contrl(2) --	0
	contrl(4) --	Length of intin = 10
	intin --	Initial defaults (for example, linestyle color and character size)
	intin(1) --	Workstation identifier (device driver id). This value is used to determine which device driver to dynamically load into memory.
	intin(2) --	Linetype
	intin(3) --	Polyline color index
	intin(4) --	Marker type
	intin(5) --	Polymarker color index
	intin(6) --	Text font
	intin(7) --	Text color index
	intin(8) --	Fill interior style
	intin(9) --	Fill style index
	intin(10) --	Fill color index
Output	contrl(3) --	Number of output vertices = 6
	contrl(5) --	Length of intout = 45
	intout(1) --	Maximum addressable width of screen/plotter in rasters/ steps assuming a 0 start point (for example, a resolution of 640 implies an addressable area of 0-639, so intout(1)=639)
	intout(2) --	Maximum addressable height of screen/plotter in rasters/ steps assuming a 0 start point (for example, a resolution of 480 implies an addressable area of 0-479, so intout(2)=479)
	intout(3) --	Device Coordinate units flag 0 = Device capable of producing precisely scaled image (typically plotters and printers) 1 = Device not capable of precisely scaled image (CRTs)
	intout(4) --	Width of one pixel (plotter step, or aspect ratio for CRT) in micrometers
	intout(5) --	Height of one pixel (plotter step, or aspect ratio for CRT) in micrometers

intout(6) -- Number of character heights

0 = continuous scaling

intout(7) -- Number of linetypes
intout(8) -- Number of line widths
intout(9) -- Number of marker types
intout(10) -- Number of marker sizes
intout(11) -- Number of fonts intout(12) -- Number of patterns
intout(13) -- Number of hatch styles
intout(14) -- Number of predefined colors (must be at least 2 even for monochrome device). This is the number of colors that can be displayed on the device simultaneously.
intout(15) -- Number of Generalized Drawing Primitives (GDPs)
intout(16)-

intout(25) -- Linear list of GDP numbers supported -1 no more GDPs in list. Application should search list until finding a -1 for the desired GDP.

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

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

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

intout(36) -- Color capability flag

0 -- no
1 -- yes

intout(37) -- Text rotation capability flag

0 -- no
1 -- yes

intout(38) -- Fill area capability flag

0 -- no
1 -- yes

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

0 -- no
1 -- yes

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

0 -- continuous device
(morethan32767 colors)
2 -- monochrome (black and white)
›2 -- number of colors available

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

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

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

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

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

Monochrome CRT type devices

Index	Color
0	Black
1	White

Monochrome Printer/Plotter devices

Index	Color
0	White
1	Black

Color

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

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

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

Description

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

CLOSE WORKSTATION Stop all graphics output to this workstation.

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

Output
contrl(3) -- 0

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

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

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

Output
contrl(3) -- 0

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

UPDATE WORKSTATION Display all pending graphics on workstation.

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE Perform device specific operation.

Input

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

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

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

Description

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

ESCAPE: INQUIRE ADDRESSABLE CHARACTER CELLS

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

Input

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

Output

contrl(3) -- 0

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

Description

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

ESCAPE: ENTER GRAPHICS MODE

Enter graphics mode if different from alpha mode.

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: EXIT GRAPHICS MODE

Exit graphics mode if different from alpha mode.

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: CURSOR UP

Move alpha cursor up one row without altering horizontal position.

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: CURSOR DOWN

Move alpha cursor down one row without altering horizontal position.

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: CURSOR RIGHT

Move alpha cursor right one column without altering vertical position.

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: CURSOR LEFT

Move alpha cursor left one column without altering vertical position.

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: HOME CURSOR

Send cursor to home position.

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: ERASE TO END OF SCREEN

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

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: ERASE TO END OF LINE

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

Input

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

Output

contrl(3) -- 0

Description

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

ESCAPE: DIRECT CURSOR ADDRESS

Move alpha cursor to specified row and column.

Input

contrl(2) -- 0
contrl(6) -- Function id = 11

intin(1) -- Row number (1 - number of rows)
intin(2) -- Column number (1 - number of columns)

Output

contrl(3) -- 0

Description

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

ESCAPE: **OUTPUT CURSOR** **ADDRESSABLE TEXT**	Output text at the current alpha cursor position.
Input	contrl(2) -- 0 contrl(4) -- Number of characters in character string contrl(6) -- Function id = 12 intin -- Text string in ASCII
Output	contrl(3) -- 0
Description	This operation displays a string of text starting at the current cursor position. Alpha text characteristics are determined by the attributes currently in effect (for example, reverse video).

ESCAPE: REVERSE VIDEO ON	Display subsequent cursor addressable text in reverse video.
Input	contrl(2) -- 0 contrl(6) -- Function id = 13
Output	contrl(3) -- 0
Description	This operation causes all subsequent text to be displayed in reverse video format; that is, characters are dark on a light background.

ESCAPE: REVERSE VIDEO OFF	Display subsequent cursor addressable text in standard video.
Input	contrl(2) -- 0 contrl(6) -- Function id = 14
Output	contrl(3) -- 0
Description	This operation causes all subsequent text to be displayed in normal video format; that is, characters are light on a dark background.

ESCAPE: INQUIRE CURRENT CURSOR ADDRESS

Return the current cursor position.

Input

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

Output

contrl(3) -- 0

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

Description

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

ESCAPE: INQUIRE TABLET STATUS

Return tablet status.

Input

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

Output

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

0 = tablet not available
1 = tablet available

Description

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

ESCAPE: HARD COPY	Generate hardcopy.
Input	contrl(2) -- 0 contrl(6) -- Function id = 17
Output	contrl(3) -- 0
Description	This operation causes the device to generate a hardcopy. This function is very device specific and can entail copying the screen to a printer or other attached hardcopy device.

ESCAPE: PLACE GRAPHIC CURSOR AT LOCATION	Place a graphic cursor at specified location
Input	contrl(2) -- 2 contrl(6) -- Function id = 18 ptsin(1) -- x-coordinate of location to place cursor ptsin(2) -- y-coordinate of location to place cursor
Output	contrl(3) -- 0
Description	Place Graphic Cursor at the specified location. This is device dependent and can be an underbar, block, or similar character. This cursor should be the same type as used for request mode locator input. In this way, if sample mode input is supported, the application may use this call to generate the cursor for rubber band type drawing. In memory mapped devices, it is drawn in XOR mode so that it can be removed. The cursor has no attributes; for example, style or color index.

ESCAPE: REMOVE LAST GRAPHIC CURSOR	Remove last graphic cursor/marker.
Input	contrl(2) -- 0 contrl(6) -- Function id = 19
Output	contrl(3) -- 0
Description	This operation removes the last graphic cursor placed on the screen.

POLYLINE Output a polyine to device.

Input

contrl(1) -- Opcode = 6
contrl(2) -- Number of vertices (x,y pairs) in polyline (n)
ptsin -- Array of coordinates of polyline in device units (for example, rasters and plotter steps)
ptsin(1) -- x-coordinate of first point
ptsin(2) -- y-coordinate of first point
ptsin(3) -- x-coordinate of second point
ptsin(4) -- y-coordinate of second point
.
.
ptsin(2n-1) -- x-coordinate of last point
ptsin(2n) -- y-coordinate of last point

Output

contrl(3) -- 0

Description

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

POLYMARKER Output markers to the device.

Input

contrl(1) -- Opcode = 7
contrl(2) -- Number of markers
ptsin -- Array of coordinates in device units (n) (for example, rasters and plotter steps)
ptsin(1) -- x-coordinate of first marker
ptsin(2) -- y-coordinate of first marker
ptsin(3) -- x-coordinate of second marker
ptsin(4) -- y-coordinate of second marker
.
.
ptsin(2n-1) -- x-coordinate of last marker
ptsin(2n) -- y-coordinate of last marker

Output

contrl(3) -- 0

Description

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

TEXT	Write text at specified position.
Input	contrl(1) -- Opcode = 8 contrl(2) -- Number of vertices = 1 contrl(4) -- Number of characters in text string intin -- Word character string in ASCII units ptsin(2) -- y-coordinate of start point of text in device units
Output	contrl(3) -- 0
Description	This operation writes text to the display surface starting at the position specified by the input parameters. Note that the X,Y position specified is the lower left corner of the character itself, not the character cell. Also, make sure the text exhibits current text attributes: color, height, character up vector, font. Each word of the intin array contains only one character. Any character code out of range for the selected font should be mapped to a blank.

FILLED AREA Fill a polygon.

Input

contrl(1) -- Opcode = 9
contrl(2) -- Number of vertices in polygon (n)
ptsin -- Array of coordinates of polygon in device units
ptsin(1) -- x-coordinate of first point
ptsin(2) -- y-coordinate of first point
ptsin(3) -- x-coordinate of second point
ptsin(4) -- y-coordinate of second point
.
.
ptsin(2n-1) -- x-coordinate of last point
ptsin(2n) -- y-coordinate of last point

Output

contrl(3) -- 0

Description

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

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

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

CELL ARRAY Display cell array.

Input

contrl(1) -- Opcode = 10
contrl(2) -- 2
contrl(4) -- Length of color index array
contrl(6) -- Length of each row in color index array (size as declared in a high level language)

contrl(7) -- Number of elements used in each row of color index array

contrl(8) -- Number of rows in color index array
contrl(9) -- Pixel operation to be performed

1 -- replace
2 -- overstrike
3 -- complement (xor)
4 -- erase

intin(1) -- Color index array (stored one row at time)
ptsin(1) -- x-coordinate of lower left corner in device units
ptsin(2) -- y-coordinate of lower left corner in device units
ptsin(3) -- x-coordinate of upper right corner in device units
ptsin(4) -- y-coordinate of upper right corner in device units

Output

contrl(3) -- 0

Description

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

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

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

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

GENERALIZED DRAWING PRIMITIVE (GDP)

Output a primitive display element.

Input

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

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

ptsin -- Array of coordinates for GDP

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

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

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

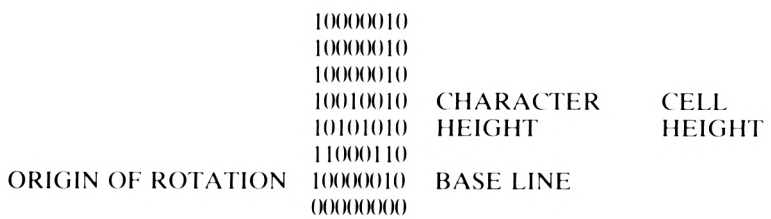
Description

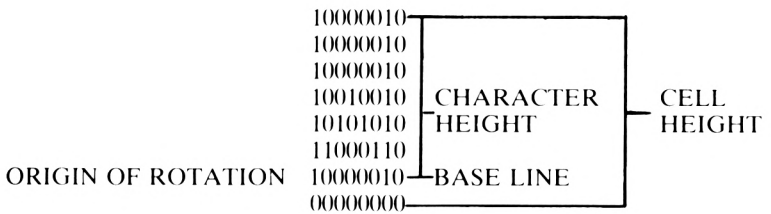
The Generalized Drawing Primitive (GDP) operation allows you to take advantage of the intrinsic drawing capabilities of your graphics device. Special elements such as arcs and circles can be accessed through this mechanism. Several primitive identifiers are predefined and others are available for expansion.

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

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

SET CHARACTER HEIGHT	Set character height.
Input	contrl(1) -- Opcode = 12 contrl(2) -- Number of vertices = 1 ptsin(1) -- 0 ptsin(2) -- Requested character height in device units (rasters, plotter steps)
Output	contrl(3) -- Number of vertices = 2 ptsout(1) -- Actual character width selected in device units ptsout(2) -- Actual character height selected in device units ptsout(3) -- Character cell width in device units ptsout(4) -- Character cell height in device units
Description	This operation sets the current text character height in Device Units. The specified height is the height of the character itself rather than the character cell. The driver returns the size of both the character and the character cell. The character size is defined as the size of an uppercase W. If the requested size does not exist, a smaller size should be used.





SET CHARACTER UP VECTOR Set text direction.

Input

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

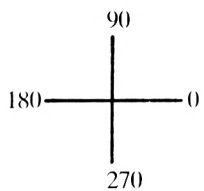
Output

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

Description

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

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



SET COLOR REPRESENTATION	Specify color index value.
Input	contrl(1) -- Opcode = 14 contrl(2) -- 0 intin(1) -- Color index intin(2) -- Red color intensity (in tenths of percent 0-1000) intin(3) -- Green color intensity intin(4) -- Blue color intensity
Output	contrl(3) -- 0
Description	This operation associates a color index with the color specified in RGB units. At least two color indexes are required (black and white for monochrome). On a monochrome device, any percentage of color should be mapped to white. On color devices without palettes, a simple remapping of the color indexes is sufficient. On color devices with palettes, loading the palette map is the proper operation. If the color index requested is out of range, no operation is performed.

SET POLYLINE LINETYPE

Set polyline linetype.

Input

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

Output

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

Description

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

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

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

SET POLYLINE LINE WIDTH — Set polyline line width.

Input

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

Output

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

Description

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

SET POLYLINE COLOR INDEX — Set polyline color index.

Input

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

Output

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

Description

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

SET POLYMARKER TYPE — Set polymarker type.

Input

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

Output

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

Description

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

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

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

SET POLYMARKER SCALE	Set polymarker scale (height).
Input	contrl(1) -- Opcode = 19 contrl(2) -- Number of input vertices = 1 ptsin(1) -- 0 ptsin(2) -- Requested polymarker height in device units
Output	contrl(3) -- Number of output vertices = 1 ptsout(1) -- 0 ptsout(2) -- Polymarker height selected in device units
Description	This operation requests a polymarker height for subsequent polymarker operations. The driver returns the actual height selected. If the selected height does not exist, use a smaller height.

SET POLYMARKER COLOR INDEX	Set polymarker color index.
Input	contrl(1) -- Opcode = 20 contrl(2) -- 0 intin(1) -- Requested polymarker color index
Output	contrl(3) -- 0 intout(1) -- Polymarker color index selected
Description	This operation sets the color index for subsequent polymarker operations. The value of the index is specified by the COLOR operation. At least two color indexes are required. If the index is out of range, use the MAXIMUM color index.

SET TEXT FONT	Set the hardware text font.
Input	contrl(1) -- Opcode = 21 contrl(2) -- 0 intin(1) -- Requested hardware text font number
Output	contrl(3) -- 0 intout(1) -- Hardware text font selected
Description	This operation selects a character font for subsequent text operations. Fonts are device-dependent and are specified from 1 to a device-dependent maximum.

SET TEXT COLOR INDEX	Set color index.
Input	contrl(1) -- Opcode = 22 contrl(2) -- 0 intin(1) -- Requested text color index
Output	contrl(3) -- 0 intout(1) -- Text color index selected
Description	This operation sets the color index for subsequent text operations. At least two color indexes are required. Color indexes range from 0 to a device-dependent maximum. If the selected index is out of range, use the MAXIMUM index.

SET FILL INTERIOR STYLE

Set interior fill style.

Input

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

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

Output

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

Description

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

SET FILL STYLE INDEX

Set fill style index.

Input

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

Output

contrl(3) -- 0

intout(1) -- fill style index selected for Pattern or Hatch fill

Description

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

1 -- vertical lines
2 -- horizontal lines
3 -- $+45^{o}$ lines
4 -- -45^{o} lines
5 -- cross
6 -- X
›6 -- device-dependent

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

SET FILL COLOR INDEX	Set fill color index.
Input	contrl(1) -- Opcode = 25 contrl(2) -- 0 intin(1) -- Requested fill color index
Output	contrl(3) -- 0 intout(1) -- Fill color index selected
Description	This operation sets the color index for subsequent polygon fill operations. The actual RGB value of the color index is determined by the SET-COLOR-REPRESENTATION operation. At least two color indexes are required. Color indexes range from 0 to a device-dependent maximum. If the selected index is out of range, use the MAXIMUM.

INQUIRE COLOR REPRESENTATION

Return color representation.

Input

contrl(1) -- Opcode = 26
contrl(2) -- 0
intin(1) -- Requested color index
intin(2) -- Set or realized flag

0 = set (return color values requested)
1 = realized (return color values realized on device)

Output

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

Description

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

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

INQUIRE CELL ARRAY Return cell array definition.

Input

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

Output

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

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

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

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

Description

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

INPUT LOCATOR Return locator position.

For REQUEST MODE

Input

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

1 = keyboard
2 = mouse, joystick

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

Output

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

0 = request unsuccessful
›0 = request successful

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

Description for Request Mode

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

For SAMPLE MODE

Input

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

1 = keyboard
2 = mouse, joystick

Output

Table C-1. Sample Mode Status Returned

Event	ControlArray (3)	(5)
Coordinates Change	1	0
key Pressed; Coordinates Not Changed	0	1
No Input	0	1

Output

contrl(3) -- Number of output vertices

1 = coordinate changed
0 = no coordinate changed

contrl(5) -- Length of intout array

0 = no terminating character
1 = terminating character returned

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

ptsout -- Returned if coordinate changed

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

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

Description for Sample Mode

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

INPUT VALUATOR Return value of valuator device.

For REQUEST MODE

Input

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

Output

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

intout(1) -- Output value

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

Description for Request Mode

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

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

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

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

For SAMPLE MODE

Input

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

Output

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

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

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

Description for Sample Mode

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

INPUT CHOICE
For REQUEST MODE

Return choice device status keys.

Input

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

1 = function keys

> 1 = workstation-dependent

Output

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

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

Description for Request Mode

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

For SAMPLE MODE

Input

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

1 = function keys

> 1 = workstation-dependent

Output

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

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

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

Description for Sample Mode

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

INPUT STRING Return string from specified string device.

For REQUEST MODE

Input

contrl(1) -- Opcode = 31
contrl(2) -- 0 if nonecho mode 1 if echo mode
intin(1) -- String device number
1 = default string device (keyboard)
intin(2) -- Maximum string length
intin(3) -- Echo mode

0 = do not echo input characters
1 = echo input characters

ptsin(1) -- x coordinate of echo area in echo mode
ptsin(2) -- y coordinate of echo area in echo mode

Output

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

0 = request unsuccessful
>0 = request successful

intout -- Output string

Description for Request Mode

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

For SAMPLE MODE

Input

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

1 = default string device (keyboard)

intin(2) -- Maximum string length

Output

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

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

intout -- Output string if sample successful

Description for Sample Mode

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

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

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

SET WRITING MODE Set writing mode

Input

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

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

Output

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

Description

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

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

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

REPLACE MODE	Replace mode is insensitive to the currently displayed image. Any information already displayed is completely replaced. The mask refers to the line style or fill pattern.
Boolean Expression	NEW = (FORE and MASK) or (BACK and not MASK)
TRANSPARENT MODE	Transparent mode only affects the pixels where the mask is one and these are changed to the FORE value.
Boolean Expression	NEW = (FORE and MASK) or (OLD and not MASK)
XOR MODE	XOR mode reverses the bits representing the color.
Boolean Expression	NEW = (FORE and MASK) XOR OLD
ERASE MODE	Erase mode sets the display to the currently selected background color where the mask value is one, independent of the foreground color.
Boolean Expression	(NEW = BACK and MASK) or (OLD and not MASK)

SET INPUT MODE Set input mode.

Input

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

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

intin(2) -- Input mode

1 = request
2 = sample

Output

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

Description

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

REQUIRED OPCODE FOR CRT DEVICES

The following opcodes and subfunctions are required for CRT devices:

Table C-2. Opcode for CRT Devices

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

Table C-2. (continued)

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

REQUIRED OPCODE FOR PLOTTERS AND PRINTERS

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

Table C-2. Opcode for Plotters and 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	
11	Graphic Drawing Primitive (GDP)	

Table C-2. (continued)

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

End of Appendix C

Glossary

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

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

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

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

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

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

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

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

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

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

VDI Virtual Device Interface

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

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

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

End of Glossary

Index

T

V

GSX™
Graphics Extension
Programmer's
Language Reference Manual

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

****
* First Edition: October 1983 *
****

Foreword

OBJECTIVE

The Digital Research Graphics SystemExtension (GSX™) provides your operatingsystem with adevice-independent interface for your applicationprogram. This manual, the *GSX Graphics Extension Programmer's Language Reference Manual* (Release 2.0), tells youto access the GSX from an application program written in one of the following high-levellanguages:

Pascal/MT+™
Pascal/MT+86™
PL/I
PL/I-86™
Digital Research C (Small Memory Model)
Digital Research C (Large Memory Model)
CBASIC Compiler (CB86..)
Digital Research FORTRAN-77™
(Small Memory Model)
Digital Research FORTRAN-77
(Large Memory Model)

Refer to the *GSX Graphics Extension User's Guide* for GSX set-up procedures. The *GSX Graphics Extension Programmer's Guide* describes components of GSX: the Graphics Device Operating System (GDOS) and the Graphics Input/Output System (GIOS).

ORGANIZATION

This manual is divided into six sections. Section 1 introduces you to accessing GSX with the languages in the preceding list.

Section 2 shows you how to access GSX from Pascal/MT+ and Pascal/MT+86. Section 3 describes how to access GSX from PL/I and PL/I-86. Section 4 describes accessing GSX from Digital Research C in both the Small and Large Models. Section 5 describes how to access GSX from CBASIC Compiler. Section 6 describes how to access GSX from Digital Research FORTRAN-77 in both the Small and Large Models.

Table of Contents

Table of Contents (continued)

Table, Figures, and Listings

Table, Figures, and Listings (continued)

Section 1

OVERVIEW

INTRODUCTION

This document is for you if you write graphics applications in a Virtual Device Interface (VDI) environment using Digital Research's Graphics System Extension (GSX). You can develop your programs in an output device-independent environment with GSX.

This manual includes the following for each language:

- GSX parameter list array addresses
- calling sequences
- array declarations
- assembly language interface routines
- example of a source listing

You can compile and link your high-level language application programs in the usual manner. Programs written in Pascal/MT+, PL/I, and CBASIC can be compiled in CP/M®, CP/M-86®, Concurrent CP/M, and PC DOS operating systems without source modification. Programs written in Digital Research C and Digital Research FORTRAN-77 can be compiled in the CP/M-86 operating system without source modification.

The assembly language routines supplied, support the GSX calling conventions and interface to your high-level language with appropriate calls to the Graphics Device Operating System (GDOS).

The example programs demonstrate how the GSX functions can be easily accessed from the assembly language interface routines to draw lines, insert text, and so on.

DISTRIBUTION FILES Table 1-1 lists the files included on your distribution disk.

Table 1-1. Distribution Files

Filename	Contents
Assembly Routines and their Object Modules	
GSXPASCL.I86	Pascal/MT+86 to GSX-86™
GSXPLI.A86	PL/I-86 to GSX-86
GSXSMALL.A86	Digital Research C to GSX-86 for C programs compiled as Small Memory Model
GSXLARGE.A86	Digital Research C to GSX-86 for C programs compiled as Large Memory Model
GSXCB80.ASM	CB80 to GSX-80™
GSXCB86.A86	CB86 to GSX-86
GSXPASCL.ASM	Pascal/MT+80 to GSX-80
GSXPLI.ASM	PL/I-80 to GSX-80
GSXSMALL.A86	FORTRAN-77 to GSX-86 for FORTRAN-77 programs compiled as Small Memory Model
GSXLARGE.A86	FORTRAN-77 to GSX-86 for FORTRAN-77 programs compiled as Large Memory Model
Executable Program Files	
DEMOP.CMD	Pascal/MT+
DEMOPLI.CMD	PL/I
DEMOC.CMD	Digital Research C
TESTFILL.CMD	8086 Assembly
Sample Program Source Files	
DEMOP.SRC	Pascal/MT+
DEMOPLI.PLI	PL/I
DEMOC.C	Digital Research C
DEMOCB.BAS	CBASIC ®
DEMOF.F77	FORTRAN-77
GSXDATA.F77	FORTRAN-77
TESTFILL.A86	8086 Assembly

Section 2
Pascal/MT+

INTRODUCTION

This section tells you how to access GSX from Pascal/MT+. This section includes listings of the assembly language interface routines to GSX from Pascal/MT+80 and from Pascal/MT+86. A sample source listing follows the assembly language interface routines. Refer to the *Pascal/MT+ Language Reference Manual* and the *Pascal/MT+ Language Programmer's Guide* for more information about programming with Pascal/MT+.

The assembly language interface routine to GSX allows users to make graphics calls directly from their Pascal/MT+ programs. The sample Pascal/MT+ program and its graphics output show how to use this interface module.

The interface routine sets the GDOS function code 0473 (HEX) in register CX, and the pointer to the parameter list arrays in DS:DX before it calls your disk operating system. Refer to the VDI specification in the *GSX Graphics Extension Programmer's Guide* for a detailed explanation of the parameter lists.

ARGUMENT PASSING

Ptsout, intout, ptsin, intin, and contrl are the five standard GSX parameter list array addresses. When the application program calls the GSX interface routine from Pascal/MT+, the following is the calling sequence:

```
GSX (ptsout, intout, ptsin, intin, contrl) ;
```

Pascal/MT+ parameters are passed on the stack. Upon entry to the GSX assembly language interface routine, the top of the stack contains the return address to the example program, followed by the parameters stored in REVERSE order from the calling sequence. The procedure is declared as EXTERNAL in the Pascal/MT+ program and as PUBLIC in the assembly language interface module. The assembly language interface routine builds a parameter block from the values on the stack and passes the address of the parameter block to the GDOS.

Pascal/MT+80

The stack upon entry to the interface module is as follows:

Pascal/MT+80: one word addresses are passed

STACK :
+0 Return Address
+2 Address of Control Array
+4 Address of Integer Input Array
+6 Address of Point Input Array
+8 Address of Integer Output Array
+10 Address of Point Output Array

Pascal/MT+86

The stack upon entry to the interface module is as follows:

Pascal/MT+86: two word addresses are passed (segment:offset)

STACK :
+0 Return Address
+4 Address of Control Array
+8 Address of Integer Array
+12 Address of Point Input Array
+16 Address of Integer Output Array
+20 Address of Point Output Array

ARRAY DECLARATION Declare the GSX parameter list arrays as shown below. The size of these parameter arrays depends on your application memory requirements.

```
type
    contrl_array : array [1..length_contrl ] of integer ;
    intin_array : array [1..length_intin ] of integer ;
    ptsin_array : array [1..length_ptsin ] of integer ;
    intout_array : array [1..length_intout ] of integer ;
    ptsout_array : array [1..length_ptsout ] of integer ;
External Procedure GSX (var ptsout: ptsout_array ;
                        var intout: intout_array ;
                        var ptsin: ptsin_array ;
                        var intin: intin_array ;
```

Note: The assembly interface module removes all parameters from the stack before returning to the calling routine.

INTERFACE ROUTINES FROM PASCAL/MT+ The following are the assembly interface routines to GSX from Pascal/MT+80 and Pascal/MT+86.

Listing 2-1. Pascal/MT+80 to GSX-80 Assembly Language Interface Routine

```
;*****************************************************************
;                                                                 *
;     INTERFACE to GSX-80 from PASCAL/MTPLUS                      *
;                                                                 *
;     File Name :     GSXPASCL.ASM (RMAC Source)                  *
;                                                                 *
;     Calling Sequence :                                          *
;          GSX ptsout, intout, ptsin, intin, contrl)              *
;               arrays are passed as var params                   *
;          NOTE the order of the calling sequence                 *
;                                                                 *
;     Entry:          return address on Stack+0                   *
;                     pointer to array contrl on Stack+2          *
;                     pointer to array intin on Stack+4           *
;                     pointer to array ptsin on Stack+6           *
;                     pointer to array intout on Stack+8          *
;                     pointer to array ptsout on Stack+10         *
;                                                                 *
;     Exit:           return address on stack                     *
;                                                                 *
;     Notes :         calls bdos with ‹DE› points at the location of *
;                     param block in stack                        *
;                                                                 *
;*****************************************************************
        public      GSX
BDOS    equ         5
GSX:
        lxi         h,2;        Skip the return address on stack
        dad         sp          ; Get pointer to the parameter address
        xchg                    ; into DE
        mvi         c,73H       ; GSX Function number
        call        BDOS        ; Call BDOS

        pop         d           ; get return address
        lxi         h,10
        dad         sp          ; clean up stack
        sphl
        xchg                    ; return address into HL
        pchl

        end
```

Listing 2-2.Pascal/MT+86 to GSX-86 Assembly Language Interface Routine

```
;*****************************************************************
;                                                                 *
;     INTERFACE TO GSX-86 FROM PASCAL/MT+86                       *
;                                                                 *
;     File name: GSXPASCL.I86      (ASMT-86 Source)               *
;                                                                 *
;     Calling sequence :                                          *
;                     GSX (ptsout, intout, ptsin, intin, contrl)  *
;                           arrays are passed as var params       *
;                                                                 *
;     Entry :         return address on stack+0                   *
;                     pointer to array contrl on stack+2          *
;                     pointer to array intin on stack+6           *
;                     pointer to array ptsin on stack+10          *
;                     pointer to array intout on stack+14         *
;                     pointer to array ptsout on stack+18         *
;                                                                 *
;     Exit :          return address on stack                     *
;                                                                 *
;     Notes :         calls bdos with DS:DX points at the location of *
;                     parameter block on stack                    *
;                                                                 *
;     Parameter Arrays Declared in Pascal program :               *
;                     type                                        *
;                     param_array : array [ 1..Array_size] of integer *
;                     external                                    *
;                     procedure GSX( var ptsout, intout, ptsin, intin, *
;                          contrl:param_array);                   *
;                                                                 *
;     Rather than move data off the stack, this routine points    *
;     the data segment at the stack and dx at the position of the *
;     parameter block in the stack frame. Parameters are passed in *
;     reverse order to get them on the stack in the correct order *
;     for GDOS.                                                   *
;                                                                 *
;     History :       Sept. 28, 1983 rhk                          *
;                                                                 *
;*****************************************************************
;
        public      GSX
        name        pasgsx
        assume      cs:code, ds:data
data    segment     public
data    ends
code    segment     public
GDOS    equ         224
```

Listing 2-2. (continued)

```
GSX     proc    near
        push    ds          ; Save caller's data segment register
        push    es          ; Save caller's extra segment register
;
        mov     ax,ss       ; Move stack segment into AX
        mov     ds,ax       ; Set DS to SS
        mov     dx,sp       ; Move stack pointer into DX
        add     dx,6        ; Add 6 to point past DS, ES and ret address
        mov     cx,473h     ; GSX function number into CX
        int     GDOS        ; Call GDOS
;
        pop     es          ; Restore caller's segemtn registers
        pop     ds
        ret     20          ; return and pop 5 array addresses off stack
GSX     endp
;
code    ends
        end
```

Pascal/MT+ EXAMPLE

The following listing is a Pascal/MT+ source program that uses the assembly language interface module for GSX. Compile the file DEMOP.SRC provided on the distribution disk before you link it. Use the following commands to compile and link this file.

A>**MT+86 A:DEMOP**
Pascal/MT+86 - Release 3.0 - March 8, 1982 Copyright (c) 1982 Digital Research

Note: After compiling, you must rename your file from GSXPASCL.REL to GSXPASCL.ERL before you can link it.

A>**LINKMT A:DEMOP,GSXPASCL,PASLIB/S**
Link/MT+86..Release 3.0 - March 25, 1982 (c) Copyright 1982 by Digital Research

Note: You must run the GENGRAF utility to attach the GSX-80 loader to your program in CP/M-80 as follows:

A>**MTPLUS DEMOP**
A>**LINKMT DEMOP,GSXPASCL,PASLIB/S**
A>**GENGRAF DEMOP**

Listing 2-3. Pascal/MT+ Example Source Listing

```
(*****************************************************************
*                                                                *
* file name : demop.src                                          *
*                                                                *
* purpose   : demonstration program for PASCAL interfacing to GSX *
*             PASCAL MTPLUS and MT+86                            *
*                                                                *
*           declares five GSX parameter arrays for -             *
*           control, integer in, points in, integer out, points out *
*                                                                *
* calls     : GSX (ptsout, intout, ptsin, intin, contrl)         *
*           Note the calling order of the 5 parameter arrays     *
*                                                                *
* to link   : linkmt demop,gsxpascl,trancend,fpreals,paslib/s/d:7900 *
*                                                                *
* history   : 08 aug 1983 RK                                     *
*                                                                *
*****************************************************************)
program demop ;
(* declare global types and constants for GSX *)
const
   OPEN_CMD                = 1 ;
   CLOSE_CMD               = 2 ;
   CLEAR_CMD               = 3 ;

   PLINE_CMD               = 6 ;
   TEXT_CMD                = 8 ;
   FILAREA_CMD             = 9 ;
   GDP_CMD                 = 11 ;

   TEXT_HGT_CMD            = 12 ;

   LINE_STYL_CMD           = 15 ;
   FILL_STYL_CMD           = 23 ;
   FILL_INDX_CMD           = 24 ;

   MAX_CNTL_VALS           = 10 ;
   MAX_INTIN_VALS          = 80 ;
   MAX_INTOUT_VALS         = 45 ;
   MAX_PTS_VALS            = 100;
type
   cntrl_array= array [ 1..MAX_CNTL_VALS ] of integer ;
   intin_array = array [ 1..MAX_INTIN_VALS ] of integer;
   intout_array = array [ 1..MAX_INTOUT_VALS ] of integer;
   ptsin_array = array [ 1..MAX_PTS_VALS] of integer;
   ptsout_array = array [ 1..MAX_PTS_VALS ] of integer;
```

Listing 2-3. (continued)

```
    i_point = record
                x,y:integer
              end ; (* i-point record *)
    i_rectangle = record
                left,right,bottom,top:integer
              end ; (* i_rectangle record *)
var
    contrl :      cntrl_array;(*global arrays for calling GSX *)
    intin :       intin_array;
    intout :      intout_array;
    ptsin :       ptsin_array;
    ptsout :      ptsout_array;
    cur_dev :     integer;
    ch :          char;

(* GSX procedure tools *)

external procedure GSX(var ptsout :   ptsout_array;
                       var intout :   intout_array;
                       var ptsin :    ptsin_array;
                       var intin :    intin_array;
                       var contrl:    cntrl_array ) ;
(* draw text with current attributes *)
procedure draw_text( x, y : integer ; s : string) ;
var
    i : integer ;
begin
    contrl[ 1 ] := TEXT_CMD ;
    contrl[ 2 ] := 1 ;
    contrl[ 4 ] := length( s ) ;
    ptsin[ 1 ] := x ;
    ptsin[ 2 ] := y ;
    for i := 1 to length( s ) do
        intin[ i ] := ord( s[ i ] ) ;
    GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure show_text *)
```

Listing 2-3. (continued)

```
(* load ptsin with rectangle (5) points *)
procedure draw_rect( rect : i_rectangle );
begin
   contrl[ 1 ] := PLINE_CMD ;
   contrl[ 2 ] := 5 ;
   with rect do
         begin
         ptsin[ 1 ] := left ;
         ptsin[ 2 ] := bottom ;
         ptsin[ 3 ] := right ;
         ptsin[ 4 ] := bottom ;
         ptsin[ 5 ] := right ;
         ptsin[ 6 ] := top ;
         ptsin[ 7 ] := left ;
         ptsin[ 8 ] := top ;
         ptsin[ 9 ] := left ;
         ptsin[10 ] := bottom ;
         end ; (* with *)
   GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure draw_rect *)

(* show bar with current attributes *)
procedure Bar( rect : i_rectangle ) ;
begin
   contrl[ 1 ] := GDP_CMD ;
   contrl[ 2 ] := 2 ;                  (* 2 points in ptsin *)
   contrl[ 4 ] := 0 ;                  (* 0 points in intin *)
   contrl[ 6 ] := 1 ;                  (* 1 = bar GDP *)
   with rect do
         begin
         ptsin[ 1 ] := left ;
         ptsin[ 2 ] := bottom ;
         ptsin[ 3 ] := right ;
         ptsin[ 4 ] := top ;
         end ; (* with *)
   GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure Bar *)
```

Listing 2-3. (continued)

```
procedure draw_bar ( lowx, wide : integer ) ;
var
   i_rect : i_rectangle ;
        i : integer ;
begin
   for i := 1 to 6 do
   begin
         with i_rect do
            begin
            left := lowx ;
            bottom := lowx ;
            right := lowx + wide ;
            top := lowx + wide ;
            end ;
         set_attrib(fill_indx_cmd, i) ; (* set fill index *)
         bar( i_rect ) ;                (* bar fill *)
         lowx := lowx + (wide div 2) ;
   end ; (* for i *)
end ; (* procedure draw_bar *)

procedure set_txt_hgt( height : integer ) ;
begin
   contrl[ 1 ] := TEXT_HGT_CMD ;
   contrl[ 2 ] := 1 ;
   ptsin[ 1 ] := 0 ;
   ptsin[ 2 ] := height ;
   GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure set_txt_hgt *)

procedure draw_line(x1, y1, x2, y2 : integer ) ;
begin
   contrl[ 1 ] := PLINE_CMD ;
   contrl[ 2 ] := 2 ;
   ptsin[ 1 ] := x1 ;
   ptsin[ 2 ] := y1 ;
   ptsin[ 3 ] := x2 ;
   ptsin[ 4 ] := y2 ;
   GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure draw_line *)

procedure set_attrib( cmd, attribute : integer ) ;   (* use something else if *)
begin                                                (* want to get set style *)
   contrl[ 1 ] := cmd ;                              (* back from GSX *)
   contrl[ 2 ] := 0 ;
   intin[ 1 ] := attribute ;
   GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure set_attrib *)
```

Listing 2-3. (continued)

```
procedure draw_border ;
var
   i_rect : i_rectangle ;
begin
   with i_rect do
      begin
      left := 0 ;
      bottom := 0 ;
      right := 32767 ;
      top := 32767 ;
      end ;
   Draw_rect( i_rect ) ;              (* rectangle around screen limits *)
end ; (* procedure draw_border *)

procedure exit_gsx ;
begin
   contrl[ 1 ] := CLOSE_CMD ;
   contrl[ 2 ] := 0 ;
   GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure exit_gsx *)

procedure clear_it ;
begin
   contrl[ 1 ] := CLEAR_CMD ;
   contrl[ 2 ] := 0 ;
   GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure clear_it *)

procedure open_wk( dev_no : integer ) ;
var
   i : integer ;                      (* to load intin with defaults *)
 begin
   contrl[ 1 ] := OPEN-CMD ;
   contrl[ 2 ] := 0 ;                 (* no input coordinates *)
   contrl[ 4 ] := 10 ;                (* 10 integers in intin array *)
   intin[ 1 ] := dev_no ;
   for i := 2 to 10 do
   intin[ i ] := 1 ;                  (* default attributes *)
   GSX( ptsout, intout, ptsin, intin, contrl ) ;
end ; (* procedure open_wk *)
```

Listing 2-3. (continued)

```
procedure init_gsx ;
begin
   writeln( 'Which output device- ' ) ;
   writeln( '1 - CRT' ) ;
   writeln( '11 - Plotter' ) ;
   writeln( '21 - Printer' ) ;
   write( '? ' ) ;
   readln( cur_dev ) ;
   open_wk( cur_dev ) ;
end ; (* procedure init_gsx *)

begin (* main program *)
   Init_gsx ;                    (*Initialize graphics device *)
   Clear_it ;                    (* clear display screen *)
   Draw_border ;                 (* outline display area *)
   Set_attrib (line_styl_cmd, 3) ;  (* set line style *)
   Draw_line (0, 0, 32767, 32767) ; (* draw a diagonal line *)
   Set_txt_hgt (1000) ;          (* set character height *)
   Draw_text (16383, 16383, 'Pascal-GSX Demo') ;    (* output text *)
   Set_attrib (fill_styl_cmd, 2) ;  (* set fill style *)
   Draw_bar (1200, 1200) ;       (* draw bars *)
   Set_attrib (fill_styl_cmd, 3) ;  (* set fill style *)
   Draw_bar (21968, 1200) ;      (* draw bars *)
   readln (ch) ;                 (* wait for a key stroke *)
   Exit_gsx ;                    (* exit graphics *)
end. (* main program *)
```

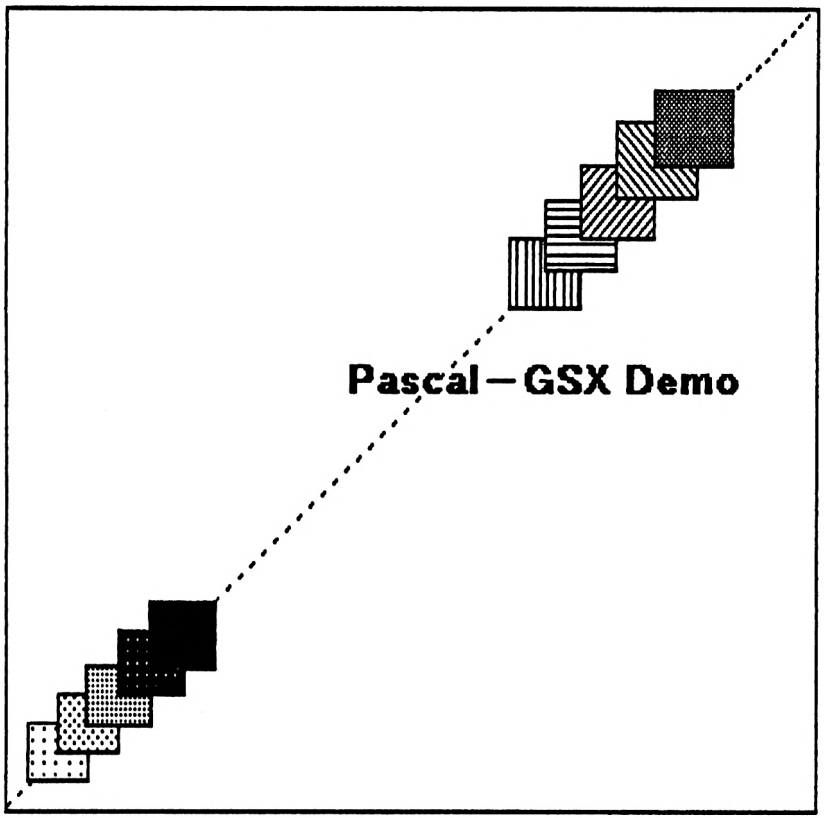


Figure 2-1. Output from the Pascal/MT+86 Example

End of Section 2

Section 3
PL/I

INTRODUCTION

This section tells you how to access GSX from PL/I and includes listings of the assembly interface routine to GSX from PL/I-80 and from PL/I-86. A sample source listing follows the assembly interface routines. Refer to the *PL/I Language Reference Manual* and the *PL/I Language Programmer's Guide* for more information about programming with PL/I.

The assembly language interface routine to GSX allows you to make graphics calls directly from PL/I programs. The sample PL/I program and its graphics output show how to use this interface module.

The interface routine sets the GDOS function code 0473 (HEX) in register CX, and the pointer to the parameter list arrays in DS:DX before it calls your disk operating system. Refer to the VDI specification in the *GSX Graphics Extension Programmer's Guide* for a detailed explanation of parameter lists.

ARGUMENT PASSING FROM PL/I TO ASSEMBLY

Contrl, intin, ptsin, intout, and ptsout are the five standard GSX parameter list array addresses. When calling the GSX assembly language interface routine from your PL/I application program, the calling sequence is the following: GSX (contrl, intin, pstin, intout, ptsout) ; PL/I-86 uses the BX register to pass parameters anywhere (the HL register in PL/I-80). Upon entry to the GSX routine, the top of the stack contains the return address, and the BX register contains an address pointer to the parameter list. This procedure is declared as EXTERNAL in the PL/I program and as PUBLIC in the assembly interface module.

The stack upon entry to the interface module is as follows:

```
STACK : +0 Return Address

‹BX› : +0 Address of Control Array
       +2 Address of Integer Input Array
       +4 Address of Point Input Array
       +6 Address of Integer Output Array
       +8 Address of Point Output Array
```

ARRAY DECLARATION

The parameter list arrays and the assembly interface modules can be declared in the following manner:

```
DECLARE
    contrl_array(length_contrl)FIXEDBIN(15);
    intin_array(length_intin)FIXEDBIN(15);
    ptsin_array(length_ptsin)FIXEDBIN(15):
    intout_array(length_intout)FIXEDBIN(15);
    ptsout_array(length_ptsout)FIXEDBIN(15);

DECLARE GSX ENTRY
    ((length_contrl)FIXEDBIN(15),
    ((length_intin)FIXEDBIN(15),
    ((length_ptsin)FIXEDBIN(15),
    ((length_intout)FIXEDBIN(15),
    ((length_ptsout)FIXEDBIN(15))EXTERNAL;
```

INTERFACE ROUTINE

The following listing is the assembly language interface routine for PL/I-80 to GSX-80.

Listing 3-1. PL/I-80 to GSX-80 Assembly Language Interface Routine

```
;*****************************************************************
;                                                                 *
;  INTERFACE TO GSX-80 FROM PL/I-80                               *
;                                                                 *
;  File Name: GSXPLI.ASM        (RMAC Source)                     *
;                                                                 *
;  PL/I calling sequence :                                        *
;          call GSX (contrl, intin, ptsin, intout, ptsout) ;      *
;                                                                 *
;  Entry :  return address on stack+0                             *
;           pointer to array contrl in ‹HL›                       *
;           pointer to array intin in ‹HL› + 2                    *
;           pointer to array ptsin in ‹HL› + 4                    *
;           pointer to array intout in ‹HL› + 6                   *
;           pointer to array ptsout in ‹HL› + 8                   *
;                                                                 *
;  Exit :   return address on stack                               *
;                                                                 *
;  Notes :  calls bdos with ‹DE› points at the location of        *
;           parameter block                                       *
;                                                                 *
;  Parameter Arrays Declared in PL/I program :                    *
;           DECLARE param_array(array_length) FIXED BIN (15) ;    *
;           DECLARE GSX ENTRY                                     *
;                       ((contrl_size) FIXED BIN(15)              *
;                       (intin_size)    FIXED BIN(15)             *
;                       (ptsin_size,2)  FIXED BIN(15)             *
;                       (intout_size)   FIXED BIN(15)             *
;                       (ptsout_size,2) FIXED BIN(15)) EXTERNAL ; *
;                                                                 *
;  Set up the GSX parameter array address pointer in ‹DE›         *
;  and calls GDOS                                                 *
;                                                                 *
;  History :            Sept. 28, 1983 rhk                        *
;                                                                 *
;*****************************************************************
        cseg
        public      GSX

BDOS    equ         5
GSX :
        xchg                    ; Address pointer to param block in DE
        mvi         c,115       ; load GSX function number
        call        BDOS        ; Call BDOS
        ret
        end
```

Listing 3-2. PL/I-86 to GSX-86 Assembly Language Interface Routine

```
;*****************************************************************
;                                                                 *
;     INTERFACE TO GSX-86 FROM PL/I-86                            *
;                                                                 *
;     File Name: GSXPLI.A86      (RASM86 Source)                  *
;                                                                 *
;     PL/I calling sequence :                                     *
;          call GSX(contrl, intin, ptsin, intout, ptsout) ;       *
;                                                                 *
;     Entry :          return address on stack+0                  *
;                      pointer to array contrl in ‹BX›            *
;                      pointer to array intin in ‹BX› + 2         *
;                      pointer to array ptsin in ‹BX› + 4         *
;                      pointer to array intout in ‹BX› + 6        *
;                      pointer to array ptsout in ‹BX› + 8        *
;                                                                 *
;     Exit :           return address on stack                    *
;                                                                 *
;     Notes :          calls bdos with DS:DX points at the location of *
;                      parameter block                            *
;                                                                 *
;     Parameter Arrays Declared in PL/I program :                 *
;          DECLARE param array (array-length) FIXED BIN (15) ;    *
;          DECLARE Gsx ENTRY                                      *
;                      ((contrl_size)   FIXED BIN (15),           *
;                      (intin_size)     FIXED BIN (15),           *
;                      (ptsin_size,2)   FIXED BIN (15),           *
;                      (intout_size)    FIXED BIN (15),           *
;                      (ptsout_size,2) FIXED BIN (15)) EXTERNAL ; *
;     Set up the data segment and the offset address for the GSX  *
;     parameter block on the stack and calls GDOS                 *
;     History :        Sept. 28,1983 rhk                          *
;                                                                 *
;*****************************************************************
        cseg
        public      GSX

        GDOS        EQU   224
GSX:
        Push        bp          ; Save caller BP
        Push        si          ; Save caller SI
        Push        di          ; Save caller DI
        Push        es          ; Save caller ES

;       Setup GSX parameter block addresses on stack
;       BX points to the parameter block
```

Listing 3-2 (continued)

```
Push    ss                  ; Points out array segment address
Push    word ptr 8 (bx)     ; Points out array offset address
Push    ss                  ; Integer out array segment address
Push    word ptr 6 (bx)     ; Integer out array offset address
Push    ss                  ; Points out array segment address
Push    word ptr 4 (bx)     ; Points out array offset address
Push    ss                  ; Integer input array segment address
Push    word ptr 2 (bx)     ; Integer input array offset address
Push    ss                  ; Control array segment address
Push    word ptr 0 (bx)     ; Control array offset address

Mov     DX, SP              ; DS:DX points to GSX parameter arrays
Mov     CX, 0473h           ; GSX Function number

Int     GDOS                ; Invoke the GDOS

Add     sp, 20              ; Clean up stack
Pop     es                  ; Restore caller's registers
Pop     di
Pop     si
Pop     bp
Ret                         ; Return to the calling PL/I program

end
```

PL/I EXAMPLE

Listing 3-3 is a PL/I source file that calls the GSX assembly language interface routine. You must compile and link the PL/I source file with the following commands:

```
A>PLI DEMOPLI

PL/I-86 Compiler                          Version 1.0
Serial No. xxxx-0000-000001   All Rights Reserved
Copyright (c) 1982,1983

Compilation of: DEMOPLI

  No Error(s) in Pass 1

  No Error(s) in Pass 2

Code Size: 03CD
Data Size: 06E6

End of Compilation
```

```
A>

A> LINK86 DEMOPLI,GSXPLI

Link-86 7/15/83                          Version 1.1
Serial No. XXXX-0000-654321 All Rights Reserved
Copyright (C) 1982,1983      DigitalResearch, Inc.,

CODE 02E25
DATA 00D8C

USE FACTOR:  04%

A> TYPE ASSIGN.SYS
01 ddds180.cmd

A>GRAPHICS

GSX-86 Graphics System Extension 3 Aug 83 V1.2.
Serial No. XXXX-0000-654321   All rights reserved
Copyright (C) 1983            DigitalResearch, Inc.,

GSX-86 installed
DDDS180 .CMD is 14352 bytes long at 0472:0000
A>DEMOPLI

Which output device -
1 - CRT
11 - Plotter
21 - Printer
```

Note: For CP/M-80, compile and link with the following commands:

```
A>PLI DEMOPLI
A>LINK DEMOPLI, GSXPLI
A>GENGRAF DEMOPLI
```

Listing 3-3. PL/I Example Source Listing

```
/*************************************************************/
/*                                                           */
/* File Name          : demopli.pli                          */
/*                                                           */
/* Purpose            : sample program demonstrating PL/I interfacing*/
/*                       to GSX                              */
/*                                                           */
/* calls              : GSX(control, intin, ptsin, intout, ptsout)  */
/*                                                           */
/* to link            : link86 demopli,gsxpli                */
/*                                                           */
/* history            : 11 Aug 1983 RHK                      */
/*                                                           */
/*************************************************************/

GSX_DEMO_PLI:
        PROC OPTIONS (MAIN)

% REPLACE opcode              BY 1;
% REPLACE vertices_in         BY 2;
% REPLACE vertices_out        BY 3;
% REPLACE intin_length        BY 4;
% REPLACE intout_length       BY 5;
% REPLACE gdp_id              BY 6;

/* Define opcodes for the control array */

% REPLACE open_work_station   BY 1
% REPLACE close_work_station  BY 2
% REPLACE clear_work_station  BY 3
% REPLACE poly_line           BY 6
% REPLACE graphic_text        BY 8
% REPLACE filled_area         BY 9
% REPLACE drawing_primitive   BY 11
% REPLACE set char_height     BY 12
% REPLACE set_line_type       BY 15
% REPLACE set_fill_style      BY 23
% REPLACE set_fill_style_index BY 24
```

Listing 3-3. (continued)

```
/* Define subscripts to the intin array for the open work station call */

% REPLACE open_station_id      BY 1
% REPLACE open_line_type       BY 2
% REPLACE open_line_color      BY 3
% REPLACE open_marker_type     BY 4
% REPLACE open_marker_color    BY 5
% REPLACE open_text_font       BY 6
% REPLACE open_text_color      BY 7
% REPLACE open_fill_interior   BY 8
% REPLACE open_fill_style      BY 9
% REPLACE open_fill_color      BY 10

DECLARE control (6) FIXED BIN (15) ;
DECLARE (intin(100), ptsin(100,2), intout (100),ptsout(100,2)) FIXED BIN (15)

   DECLARE GSX ENTRY
        ((6) FIXED BIN (15)
        (100)FIXED BIN(15)
        (100,2)FIXED BIN(15)
        (100)FIXED BIN(15)
        (100,2) FIXED BIN (15) EXTERNAL

        %REPLACE                SCREEN_MIN_X by 0
                                SCREEN_MAX_X by 32767
                                SCREEN_MIN_Y by 0
                                SCREEN_MAX_Y by 32767

        DECLARE   (dev_unit_x,dev_unit_y) FIXED BIN(15)
                  C CHAR (1)
                  DEVICE_DATA (100) FIXED BIN (15)

        DECLARE   sign_on CHAR (80) VARYING
                  STATIC INITIAL ('PL/I-GSX Demo')
```

Listing 3-3 Continued

```
/********** Main Procedure    ***********************************/
/*                                                               */
      Call INIT_GSX;              /* Initialise device           */
      Call CLEAR_IT;              /* clear display surface       */
      Call DRAW_FRAME;            /* outline display area        */
      Call SET_ATTRIB(set_line_type, 3)    /* set Line Style     */
      Call DO_LINE (screen_min_x, screen_min_y);
        screen_max_x, screen_max_y)        /* draw a diagonal line */
      Call SET_TXT_HGT (1000);            /*Set character height */
      Call DO_GRAPHIC_TEXT(16384,16384,SIGN ON) ;/*output text */
      Call SET_ATTRIB(set_fill_style,3);  /*Set hatch fill style */
      Call Draw_bar (1200,1200);          /*draw bars            */
      Call SET_ATTRIB(set_fill_style,2);  /* Set pattern fill style */
      Call Draw_Bar (21968,1200);
      GET EDIT(c) (A(1));                 /*Wait for a keystroke */
      call CLOSE;                         /*Exit graphics        */
/*                                                               */
/********** End of Main program ***********************************/

DO_LINE:
        Proc(xl,yl,x2,y2);
        Dcl (xl,yl,x2,y2) FIXED BIN(15);

        Control (opcode) = poly_line;
        /* no., of vertices in control array */
        Control (vertices_in) = 2; */ 2 input points */
        /* line vertices in points input array */
        Ptsin (1,1) = x1;
        Ptsin (1,2) = y1;
        Ptsin (2,1) = x2;
        Ptsin (2,2) = y2;
        CALL GSX(Control,intin,ptsin,intout,ptsout);
        End DO_LINE;

SET_TEXT_HGT:
        Proc(height);
        DCL     height FIXED BIN (15);

        control (opcode) = set_char_height;
        /* call GSX with character height in point input array */
        control (vertices_in) = 1;
        ptsin (1,1) = 0;
        pstin (1,2) = height;
        CALL GSX (control,intin,ptsin,intout,ptsout);

        End SET_TXT_HGT;
```

Listing 3-3 (continued)

```
DO_GRAPHIC_TEXT;
      Proc (X,Y,STRING)
      DCL    (X,Y,I) FIXED BIN(15),
             CCHAR(1),
             stringCHAR(80)VARYING;

      /* move each charactaer into integer input array */
      Do I=I to length (string);
         C = substr (string,I,1);
         intin(I) = rank (C);
         End;
      /* setup graphics text call */
      control (opcode) = graphic_text;
      /* no. of characters in integer input array */
      control (intin_length) = length (string):
      /* text position in point input array */
      control vertices_in = 1
      ptsin (1,1) = X;
      ptsin (1,2) = Y;
      CALL GSX (control, intin,ptsin, intout,ptsout);

      End DO_GRAPHIC_TEXT;

DRAW_BAR:
      Proc (X, WIDE);
      DCL (X,WIDE,LOWX,I,left,bottom,right,top) FIXED BIN (15);

      LOWX = X;
      /* draw 6 bars */
      Do I=1 to 6;
            left = LOWX;
            bottom = LOWX;
            right = LOWX + WIDE;
            top = LOWX + WIDE;
            /* set interior fill style index for next bar */
            CALL Set_attrib (set_fill_style_index,i);
            */ call gdp bar with lower left and upper right vertices */
            CALL Bar (left, bottom, right, top);
            /* update to next bar position */
            LOWX = LOWX + (WIDE / 2);
            End;

      END Draw_Bar;
```

Listing 3-3 (continued)

```
BAR:
      Proc(left,bottom,right,top);
      DCL (left,bottom,right,top) FIXED BIN (15);

      control (opcode) = drawing primitive;
      control (intin_length) = 0; /* no integer input in intin */
      control (vertices_in) = 2; /* 2 points in ptsin */
      control (gdp_id) = 1; /* gdp bar id = 1 */
      /* lower left and upper right points in points input array */
      ptsin (1,1) = left;
      ptsin (1,2) = bottom;
      ptsin (2,1) = right;
      ptsin (2,2) = top;
      CALL GSX(control, intin, ptsin, intout, ptsout);

      End BAR;

SET_ATTRIB :
      Proc (CMD,I):
      DCL (CMD,I) FIXED BIN (15);

      control (opcode) = CMD /* attribute command*/
      /* indicates no points passed */
      control (vertices_in) = 0;
      control (intin_length) = 1;
      /* pass the attribute value in integer input array */
      intin (1) = I;
      CALL GSX (control,intin,ptsin,intout,ptsout);

      END SET_ATTRIB ;

DRAW_FRAME:
      Proc;

      control(opcode) = poly_line;
      control(vertices in) = 5; /* 5 input points */
      ptsin (1,1) = screen_min_x;
      ptsin (1,2) = screen_min_y;
      ptsin (2,1) = screen_max_x;
      ptsin (2,2) = screen_min_y;
      ptsin (3,1) = screen_max_x;
      ptsin (3,2) = screen_max_y';
      ptsin (4,1) = screen_min_x;
      ptsin (4,2) = screen_max_y;
      ptsin (5,1) = screen_min_x;
      ptsin (5,2) = screen_min_y;
      CAL GSX(control,intin,ptsin,intout,ptsout);

      End DRAW_ FRAME;
```

Listing 3-3 (continued)

```
INIT_GSX:
      Proc;
         DCL I FIXED BIN (15);
         PUT SKIP LIST ('Which output device -');
         PUT SKIP LIST ('1 - CRT');
         PUT SKIP LIST ('11 - Plotter');
         PUT SKIP LIST ('21 - Plotter');
         PUT SKIP LIST ('?');
         GET LIST (i);
         Call Open (i);
      END Init_gsx;

OPEN:
      Proc(I);
      DCL I FIXED BIN (15)

      control (opcode) = open_work_station;
      /* 10 input values in integer input array */
      control (intin_length) = 10;
      /* no points passed to GSX in open workstation call */
      control (vertices_in) = 0;

      /* open the requested workstation */
      intin(open_station_id) = i;
      /* set up default parameters */
      intin (open_line_type) = 1;
      intin (open_line_color) = 1;
      intin (open_fill_interior) = 1;
      intin (open_fill_style) = 1;
      intin (open_fill_color) = 1;
      intin (open_text_color) = 1;
      intin (open_text_font) = 1;
      CALL GSX (control,intin,ptsin,device_data,ptsout);

      dev_unit_x = 32767 / device_data (1) + 1;
      dev_unit_y = 32767 / device_data (2) + 1;

      END OPEN;
CLEAR_IT:
      Proc;

      control (opcode) = clear_work_station;
      control (vertices_in) = 0;
      CALL GSX (control,intin,ptsin,intout,ptsoutf);

      END CLEAR_IT;
```

Listing 3-3 (continued)

```
CLOSE:
      Proc;

      control(opcode) = close_work_station;
      control (vertices_in) = 0;
      CALL GSX (control,intin,ptsin,intout,ptsout):

      END CLOSE

END GSX_DEMO_PLI:
```

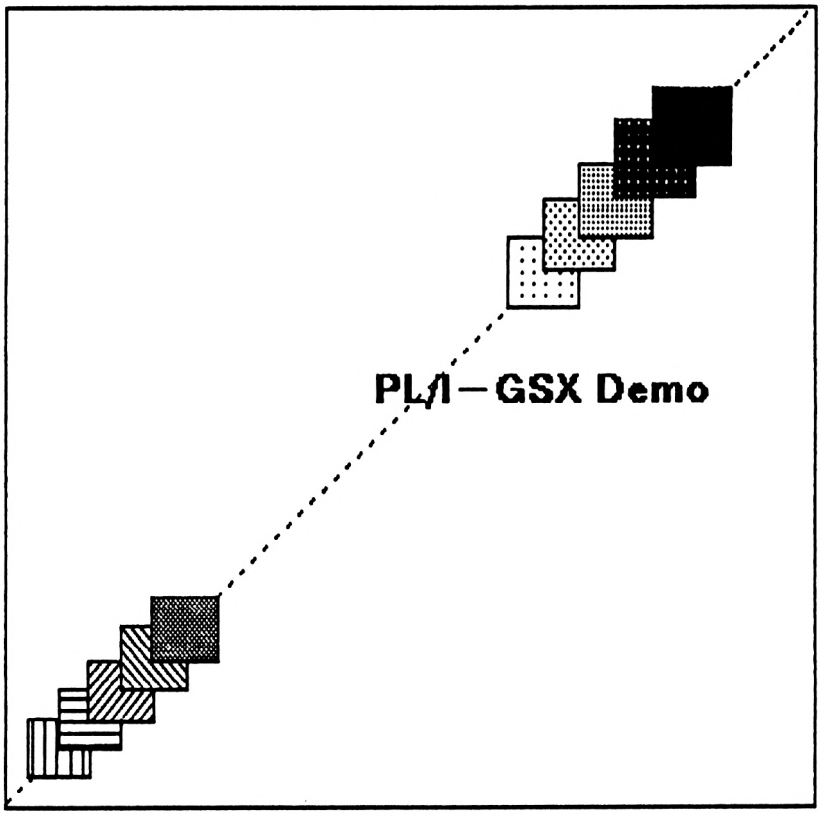


Figure 3-1. Output from the PL/I Example

Section 4
Digital Research C

INTRODUCTION

This section tells you how to access GSX from Digital Research C. Included in this section are listings of the assembly language interface routine to GSX from Small and Large Memory Models from Digital Research C. A sample source listing follows the assembly language interface routines. Refer to The C Programming Language by Kernighan and Ritchie and the C Language Programmer's Guide for the CP/M-86 Family of Operating Systems for more information about programming with Digital Research C.

The assembly language interface routine to GSX allows you to make graphics calls directly from Digital Research C programs. The sample Digital Research C program and its graphics output show how to use this interface module.

The interface routine sets the GDOS function code 0473 (HEX) in register CX, and the pointer to the parameter list arrays in DS:DX before it calls the disk operating system. Refer to the VDI specification in the GSX Graphics Extension Programmer's Guide for a detailed explanation of the parameter lists.

ARGUMENT PASSING

Contrl, intin, ptsin, intout, and ptsout are the five standard GSX parameter list array addresses. When your application program calls the GSX interface routine from C, use the following calling sequence:

```
GSX (contrl, intin, ptsin, intout, ptsout) ;
```

Parameters from the Digital Research C program are passed on the stack. Upon entry to the GSX assembly language interface routine, the top of the stack contains the return address to the example program, followed by the arguments stored in the same order as in the calling sequence. Refer to the C Language Programmer's Guide for the CP/M-86 Family of Operating Systems for more information on stacks and how address space is allocated.

For example, upon entry to the GSX interface routine from a Small Memory Model Digital Research C program, one word addresses are passed in the stack
in the following manner:

```
STACK :    +0 Return Address
           +2 Address of Control Array
           +4 Address of Integer Input Array
           +6 Address of Point Input Array
           +8 Address of Integer Output Array
           +10 Address of Point Output Array
```

ARRAY DECLARATION

The GSX parameter list arrays can be declared in the Digital Research C program as in the following example. The size of these parameter arrays depends on your application memory requirements.

```
int contrl(length_contrl);  /*  input parameters              */
int ptsin(length_ptsin);    /*  input coordinate data         */
int ptsout(length_ptsout);  /*  output coordinate data        */
int intin(length_intin);    /*  int input parameters          */
int intout(length_intout);  /*  int output parameters         */
```

INTERFACE ROUTINE

The following listing is for the Digital Research C, Small Memory Model.

Listing 4-1. Digital Research C (Small Memory Model) to GSX-86 Assembly Language Interface Routine

```
;*****************************************************************
;       INTERFACE TO GSX-86 FROM C and FORTRAN-77                *
;       (for Small Memory Model)                                 *
;                                                                *
;       File name:   GSXSMALL.A86                                *
;                                                                *
;       Calling sequence :                                       *
;                    GSX (contrl,intin,ptsin,intout,ptsout)      *
;                                                                *
;       Entry:       return address on stack-0                   *
;                    pointer to array contrl on stack+2          *
;                    pointer to array intin on stack+4           *
;                    pointer to array ptsin on stack+6           *
;                    pointer to array intout on stack+8          *
;                    pointer to array ptsout on stack+10         *
;                                                                *
;       Exit :       return address and parameters on stack      *
;                    unchanged                                   *
;                                                                *
;       Notes :      calls gdos with DS:DX points at the location *
;                    of parameter block on stack                 *
;                                                                *
;       History:              Sept. 28, 1983 rhk                 *
;                                                                *
; *****************************************************************
;
        cseg
        public  GSX

        GDOS    EQU  224
GSX:
        Push    bp                  ; Save caller BP.
        mov     bp,sp               ; Create new stack frame.
        Push    si                  ; Save Caller SI.
        Push    di                  ; Save Caller DI.
        Push    es                  ; Save Caller ES.

;       Setup GSX parameter block addresses on stack.

        Push    ss                  ; Points out array segment address
        Push    word ptr 12[bp]     ; Points out array offset address
        Push    ss                  ; Integer out array segment
                                        address
        Push    word ptr 10[bp]     ; Integer out array offset address
        Push    ss                  ; Points out array segment address
        Push    word ptr 8[bp]      ; Points out array offset address
        Push    ss                  ; Integer input array segment
                                        address
        Push    word ptr 6[bp]      ; Integer input array offset
                                        address
        Push    ss                  ; Control array segment
        Push    word ptr 4[bp]      ; Control array offset address
```

Listing 4-1. (continued)

```
Mov     DX, SP              ; DS:DX points to GSX parameter
                                arrays
Mov     CX, 0473h           ; GSX Function number

INT     GDOS                ; Invoke the GDOS

Add     sp, 20              ; Clean up stack
Pop     es                  ; Restore caller ES
Pop     di                  ; Restore caller DI
Pop     si                  ; Restore caller SI
Pop     bp                  ; Restore caller BP
Ret                         ; Return.

end
```

The following listing is the interface to GSX-86 from Digital Research C, Large Memory Model.

Listing 4-2. Digital Research C (Large Memory Model) to GSX-86 Assembly Language Interface Routine

```
;*****************************************************************
;                                                                 *
;       INTERFACE TO GSX-86 FROM C and FORTRAN-77                 *
;       (Large Memory Model)                                      *
;                                                                 *
;       File Name:   GSXLARGE.A86                                 *
;                                                                 *
;       Calling sequence :                                        *
;                    GSX (contrl, intin, ptsin, intout, ptsout)   *
;                                                                 *
;       Entry :      return address on stack+0                    *
;                    pointer to array contrl on stack+4           *
;                    pointer to array intin on stack+8            *
;                    pointer to array ptsin on stack+12           *
;                    pointer to array intout on stack+16          *
;                    pointer to array ptsout on stack+20          *
;                                                                 *
;       Exit :       return address and parameters on stack       *
;                                                                 *
;       Notes:       calls bdos with DS:DX points at the location *
;                    of parameter block address on stack          *
;                                                                 *
;       History:     Sept. 28, 1983 rhk                           *
;                                                                 *
;       This routine points the data segment at the stack         *
;       segment, and DX at the GSX parameter block on the stack   *
;       frame, returns to the calling program without moving      *
;       datas off the stack.                                      *
;                                                                 *
;                                                                 *
*****************************************************************
;
        Cseg
        Public  GSX

        GDOS    EQU 224
GSX:
        Push    ds                  ; Save caller DS.
        Push    bp                  ; Save caller BP.
        Mov     bp, sp              ; Create new stack frame.
        Push    es                  ; Save caller ES.
        Push    si                  ; Save caller SI.
        Push    di                  ; Save caller DI.

        Mov     ax, ss              ; Move stack segment into ax
        Mov     ds, ax              ; Set DS to SS
        Lea     dx, 8[bp]           ; DX points to GSX parameter
                                      array address
        Mov     cx, 0473h           ; GSX function number into cx
```

Listing 4-2. (continued)

```
Int     GDOS                    ; Call GDOS
Pop     di                      ; Restore caller DI.
Pop     si                      ; Restore caller SI.
Pop     es                      ; Restore caller ES.
Pop     bp                      ; Restore caller BP.
Pop     ds                      ; Restore caller DS.
Retf                            ; Return.

end
```

DIGITAL RESEARCH C EXAMPLE

The following listing is an example of a C source program that uses the assembly language interface routine for GSX. To use this routine you must compile and link the Digital Research C source file with the following commands:

```
A> DRC DEMOC

Digital Research C Compiler 8/08/83                    Version 1.03
Serial No. XXXX-0000-654321                  All Rights Reserved.
Copyright (c) 1983                         Digital Research,Inc.,

Digital Research C  Version 1.03 -- Preprocessor

Digital Research C  Version 1.03 -- Code Gen

democ.c:    code:1149 static     94 extern: 1000

C>
C> LINK86 DEMOC,A:GSXSMLC

LINK-86 8/02/83                                        Version1.1
Serial No. XXXX-0000-654321                  All Rights Reserved
Copyright (c) 1982,1983                    Digital Research,Inc.,

CODE 05025
DATA 01280

USE FACTOR: 12%

C>TYPE ASSIGN.SYS
21 a:ddds180.cmd
01 a:ddcrt.
```

```
C>GRAPHICS

GSX-86 Graphics System Extension                3 Aug 83 V1.2
Serial No. XXXX-0000-654321                 All Rights Reserved
Copyright (C) 1983                  Digital Research, Inc.,

GSX-86 installed
A:DDDS180 .CMD is 14352 bytes long at 0472:0000
C>
C>DEMOC
Which output device -
1 - CRT
11 - Plotter
21 - Printer
?1 -
```

Listing 4-3. Example C Source Listing

```
;*****************************************************************/
;                                                                 */
; File name :       democ.c                                       */
;                                                                 */
; Purpose :         Sample C program interfacing to GSX using     */
;                   the assembly interface module                 */
;                                                                 */
; Calls :           GSX (contrl,intin,ptsin,intout,ptsout) :      */
; **note**         Upper case character routine name in the call  */
;                                                                 */
; To Link :     link86 democ,gsxsmlc (for small memory model)     */
;                                                                 */
; history :     15 Aug 1983 rhk                                   */
;                                                                 */
;*****************************************************************/

/* define the opcodes for the control array */

#define OPEN_WORKSTATION 1
#define CLOSE_WORKSTATION 2
#define CLEAR_WORKSTATION 3
#define POLYLINE 6
#define TEXT 8
#define FILL_AREA 9
#define DRAWING_PRIM 11
#define SET_CHAR_HEIGHT 12
#define SET_LINE_TYPE 15
#define SET_FILL_STYLE 23
#define SET_STYLE_INDEX 24
```

Listing 4-3. (continued)

```
/* define the intin array subscripts for open workstation defaults*/

#define WK_ID intin(0)
#define LINE_TYPE intin(1)
#define LINE_COLOR intin(2)
#define MARKER_TYPE intin(3)
#define MARKER_COLOR intin(4)
#define TEXT_FONT intin (5)
#define TEXT_COLOR intin(6)
#define FILL_STYLE intin(7)
#define FILL_INDEX intin(8)
#define FILL_COLOR intin(9)

/*      define the subscripts to the control array */

#define OPCODE contrl(0)
#define N_PTSIN contrl(1)
#define N_PTSOUT contrl(2)
#define LEN_INTIN contrl(3)
#define LEN_INTOUT contrl(4)
#define GDP_ID contrl(5)
#define N_CHAR contrl(3)
#define TEXT_X_POS ptsin(0)
#define TEXT_Y_POS ptsin(1)

#define GDP_BAR 1
#define SOLID_LINE 1
#define DASHED_LINE 2
#define DOTTED_LINE 3
#define DASHED_DOTTED 4
#define HOLLOW_FILL 0
#define SOLID_FILL 1
#define PATTERN_FILL 2
#define HATCH_FILL 3

#define YES 1
#define NO 0
#define TRUE ! = 0
#define FALSE == 0

#define ZERO 0
#define ONE 1
#define TWO 2
#define THREE 3
#define FOUR 4
#define TEN 10
```

Listing 4-3. (continued)

```
#include <stdio.h>

int contrl[20] ;    /*        input parameters                    */
int ptsin([00] ;    /*        input coordinate data               */
int ptsout[100] ;   /*        output coordinate data              */
int intin([00] ;    /*        int input parameters                */
int intout[100] ;   /*        int output parameters               */

static int xmin = {0} ;
static int ymin = {0} ;
static int xmax = {32767} ;
static int ymax = {32767};

static int sign on[] = {'D', 'e','m','o',' ','C','-','G','S','X'} ;

main()
{
/* EXAMPLE PROGRAM demonstrating interface to GSX from C          */
/* Link with assembly interface module                            */

int c ;

/*********** Main Procedure ***********************************/

   init_gsx() ;                /* Open requested workstation      */
   clear_it () ;               /* Clear the display surface       */
   draw_frame () ;             /* Outline the display area        */
   set_attrib (SET_LINE_TYPE, DOTTED_LINE) ; /* set line style    */
   do_line (xmin,ymin,xmax,ymax) ; /* draw a diagonal line        */
   set_txt_hgt (800) ;           /* set character size            */
   do_text (16384,16384, sign on, 10) ; /* output text            */
   set_attrib (SET_FILL_STYLE, 2) ;/* pattern fill                */
   draw_bar (1200, 1200) ;       /* draw bars                     */
   set_attrib (SET_FILL_STYLE, 3) ;/* hatch full                  */
   draw_bar (21968,1200) ;
   while (!__BDOS(11,0)) ;       /* wait for a keyboard key       */
   close_it () ;                 /* Exit graphics                 */
}
/******** End of Main Program *********************************/
```

Listing 4-3. (continued)

```
/***** INIT GSX ************************************************/
; Get the graphics device id number                            */
; Open Graphics Device selected                                */

init_gsx ()
{
int device ;

   printf("Which output device -n") ;
   printf("1 - CRTn") ;
   printf("11 - Plottern") ;
   printf("21 - Printern") ;
   printf("?") ;

   scanf("%d", &device) ;

/******OPEN WORKSTATION ****************************************/

   OPCODE = OPEN_WORKSTATION ;
   N_PTSIN = ZERO ;
   LEN_INTIN = TEN ;

   WK_ID = device ;

/* intin[1] - intin[10] contains the initial default settings    */

   LINE_TYPE = SOLID_LINE ;
   LINE_COLOR = ONE ;
   MARKER_TYPE = ONE ;
   MARKER_COLOR = ONE ;
   TEXT_FONT = ONE ;
   TEXT_COLOR = ONE ;
   FILL_STYLE = HOLLOW_FILL ;
   FILL_INDEX = ONE ;
   FILL_COLOR = ONE ;

   GSX (contrl,intin,ptsin,intout,ptsout) ;
   return;
}

/******CLEAR WORKSTATION****************************************/
clear_it ()
{
   OPCODE = CLEAR_WORKSTATION ;
   N_PTSIN = ZERO ;
   GSX (contrl,intin,ptsin,intout,ptsout) ;
   return ;
} /* End of Clear workstation */
```

Listing 4-3. (continued)

```
/***** DRAW A FRAME AROUND THE DISPLAY AREA **********************/
draw_frame ()
{
   /*   use polyline to draw the box     */
   OPCODE = POLYLINE ;          /* polyline function id = 6        */
   N_PTSIN = 5 ;                /* 5 vertices passed in array ptsin */
   ptsin[8] = ptsin[6] = ptsin[0] = xmin ;
   ptsin[9] = ptsin[3] = ptsin[1] = ymin ;
   ptsin[4] = ptsin[2] = xmax ;
   ptsin[7] = ptsin[5] = ymax ;
   GSX  (contrl,intin,ptsin,intout,ptsout) ;
   return ;
   /*   End of draw frame      */

/***** SUPPORT OUTPUT PRIMITIVE ATTRIBUTE SETTINGS ***************/
set_attrib (cmd,inx)
int       cmd,inx ;
{
   OPCODE = cmd ; /* set attribute function id in control array   */
   N_PTSIN = ZERO ;
   LEN_INTIN = ONE ;
   intin[0] = inx ;             /* the index in array intin */
   GSX (contrl,intin,ptsin,intout,ptsout) ;
   return ;
} /* End of set attrib */

/***** DRAWS A SINGLE SEGMENTED LINE USING POLYLINE FUNCTION *****/
do line (x1,y1,x2,y2)
int       x1,y1,x2,y2 ;
{
   OPCODE = POLYLINE ;          /* polyline function id = 6 */
   N_PTSIN = 2 ;                /* 2 vertices */
   ptsin[0] = x1 ;
   ptsin[1] = y1 ;
   ptsin[2] = x2 ;
   ptsin[3] = y2 ;
   GSX (contrl,intin,ptsin,intout,ptsout) ;
   return ;
}
/***** SET CHARACTER HEIGHT ***************************************/
set_txt_hgt (height)
int       height;
{
   OPCODE = SET_CHAR_HEIGHT ;
   N PTSIN = ONE;
   ptsin[0] = ZERO ;                /* input x coordinates = 0     */
   ptsin[1] = height ;              /* input y coordinates = height */
   GSX (contrl,intin,ptsin,intout,ptsout) ;
   return ;
} /* End of set txt height */
```

Listing 4-3. (continued)

```
/***** OUTPUT TEXT ****************************************/
do_text (x,y,string,len)
int       x,y,string,(80),len :
{
   OPCODE = TEXT ;
   N_PTSIN = ONE ;
   N_CHAR = len ;
   TEXT_X_POS = x ;
   TEXT_Y_POS = y ;
   GSX (contrl,string,ptsin,intout,ptsout) ;
   return ;
} /* End of do text */

/***** DRAW 6 BARS USING GDP BAR FUNCTION *********************/
draw_bar (x, wide)
int       x,wide ;
{
int       i,left,bottom,right,top ;

   for (i=1; 1<7; i++)         {
        /* draw 6 bars         */
        left = bottom = x ;
        right = top = x + wide ;
        set_attrib (SET_STYLE_INDEX, i) ;
        bar (left, bottom, right, top) ;
        x = x + wide + (wide / 2) ;
   }
   return ;

} /* End draw bar */

bar (left, bottom, right, top)
int       left, bottom, right, top ;
{
   /* Use GDP BAR function */
   OPCODE = DRAWING_PRIM ;
   LEN_INTIN = ZERO ;
   N_PTSIN = TWO ;
   GDP_ID = ONE ;              /* BAR ID = 1 */
   /* lower left and upper right vertices passed in ptsin array */
   ptsin[0] = left ;
   ptsin[1] = bottom ;
   ptsin[2] = right ;
   ptsin[3] = top ;
   GSX (contrl,intin,ptsin,intout,ptsout)
   return ;
} /* End bar */
```

Listing 4-3. (continued)

```
/***** CLOSE WORKSTATION ************************************/
close_it ()
{
   OPCODE = CLOSE_WORKSTATION ;
   N_PTSIN = ZERO ;
   GSX (contrl,intin,ptsin,intout,ptsout) ;
   return ;
} /* End of close workstation */
```

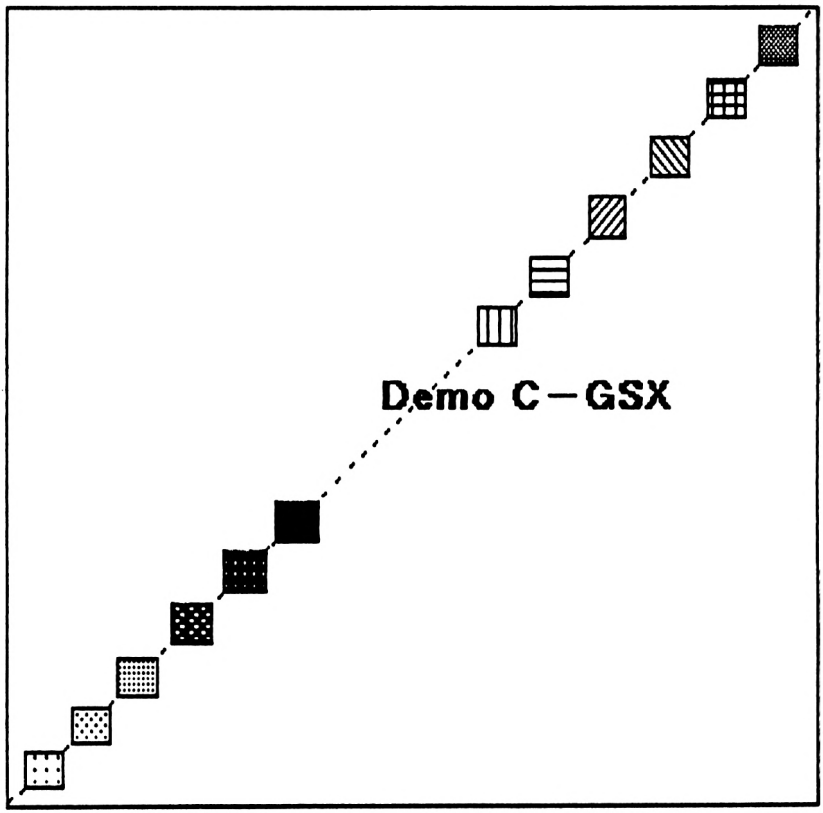


Figure 4-1. Output from the Digital Research C Example

Section 5
CBASIC COMPILER (CB86)

INTRODUCTION

Digital Research CBASIC compiler includes an extensive set of standard graphics functions. The assembly language interface routine allows you to access other GSX functions in addition to those provided in CBASIC. Refer to the CBASIC Compiler Graphics Guide for the graphics facilities supported.

ARGUMENT PASSING

Ptsout, intout, ptsin, intin, and contrl are the five standard GSX parameter list array addresses. When the application program calls the GSX interface routine from CBASIC, the calling sequence must be the following:

GSX (ptsout, intout, ptsin, intin, contrl)

CBASIC parameters are passed on the stack. Arrays are passed as VARPTR parameters. Upon entry to the GSX assembly language interface routine, the top of the stack contains the return address to the example program, followed by the parameters stored in REVERSE order from the calling sequence. The procedure is declared as EXTERNAL in the CBASIC program and as PUBLIC in the assembly language interface routine builds a parameter block from the values on the stack and passes the address of the parameter block to the GDOS.

CB80 and CB86

The stack upon entry to the interface module is as follows:

CB80 and CB86: one word addresses are passed

STACK :

+0 Return Address

+2 Address of Control Array

+4 Address of Integer Input Array

+6 Address of Point Input Array

+8 Address of Integer Output Array

+10 Address of Point Output Array

Listing 5-1. CB80 to GSX-80 Assembly Language Interface Routine

```
;*****************************************************************
;  INTERFACE to GSX-80 from CBASIC                                *
;                                                                 *
;  File Name : GSXCB80.ASM (RMAC Source)                          *
;                                                                 *
;  Calling Sequence :                                             *
;       GSX (ptsout,intout,ptsin,intin,contrl) arrays are passed as*
;          varptr parameters                                      *
;       NOTE the sequence of call parameters                      *
;                                                                 *
;  Entry: return address on Stack+0                               *
;       pointer to array contrl on Stack+2                        *
;       pointer to array intin on Stack+4                         *
;       pointer to array ptsin on Stack+6                         *
;       pointer to array intout on Stack+8                        *
;       pointer to array ptsout on Stack+10                       *
;                                                                 *
;  Exit: return address on stack                                  *
;                                                                 *
;  Notes: calls bdos with <DE> points at the location of param    *
;         block in stack                                          *
;*****************************************************************

        public GSX

BDOS    equ     5
GSX:
        lxi      h,2            ; Skip the return address on stack
        dad      sp             ; Get pointer to the parameter address
        xchg                    ; Into DE
        mvi      c,73H          ; GSX Function number into C
        call      BDOS          ; Call GDOS

        pop      d              ; Get return address
        lxi      h,10           ; Point pass the 5 parameter address
        dad      sp             ; Clean up stack
        sphl                    ; Update SP
        xchg                    ; Put return address into HL
        pchl                    ; Return

        end
```

Listing 5-2. CB86 to GSX-86 Assembly Language Interface Routine

```
; ****************************************************************
;                                                                *
;  INTERFACE TO GSX-86 FROM CB86                                 *
;                                                                *
;  File name: GSXCB86.A86                                        *
;                                                                *
;  Calling sequence :                                            *
;       GSX (ptsout,intout,ptsin,intin,contrl)                   *
;          arrays are passesd as varptr parameters               *
;  NOTE the calling sequence of the parameter list               *
;                                                                *
;  Entry : return address on stack+0                             *
;          pointer to array contrl on stack+02                   *
;          pointer to array intin on stack+04                    *
;          pointer to array ptsin on stack+06                    *
;          pointer to array intout on stack+08                   *
;          pointer to array ptsout on stack+10                   *
;                                                                *
;  Exit :       return address on stack                          *
;                                                                *
;  Notes : calls GDOS with DS:DX points at the location of       *
;          parameter block on stack                              *
;                                                                *
;  History :    Oct. 06, 1983 rhk                                *
;                                                                *
;****************************************************************
;
        cseg
        public  GSX
        GDOS    EQU  224
GSX:    Push    bp                  ; Save caller BP
        mov     bp, sp                  ; Create new stack frame
        Push    si                  ; Save caller SI
        Push    di                  ; Save caller DI
        Push    es                  ; Save caller ES

        Setup GSX parameter block addresses on stack

        Push    ss              ; Points out array segment address
        Push    word ptr 12[bp]; Points out array offset address
        Push    ss              ; Integer out array segment address
        Push    word ptr 10[bp]; Integer out array offset address
        Push    ss              ; Points out array segment address
        Push    word ptr 08[bp]; Points out array offset address
        Push    ss              ; Integer input array segment address
        Push    word ptr 06[bp]; Integer input array offset address
        Push    ss              ; Control array segment address
        Push    word ptr 04[bp]; Control array offset address
```

Listing 5-2. (continued)

```
Mov   DX, SP          ; DS:DX points to GSX parameter arrays
Mov   CX, 0473h ; GSX Function number

INT   GDOS          ; Invoke the GDOS
Add   sp, 20          ; Clean up stack
Pop   es              ; Restore ES
Pop   di              ; Restore DI
Pop   si              ; Restore SI
Pop   bp              ; Restore BP
Cld                   ; Clear direction flag
Ret   10              ; Clean up stack, return to caller

end
```

Listing 5-3. CB86 Example Program

```
REM*************************************************************
REM*
REM*    file name : DEMOCB.BAS
REM*
REM*    purpose : demonstration program for CBASIC Interfacing
REM*                  to GSX
REM*       declares five GSX parameter arrays for -
REM*       control, integer in, points in, integer out, points out
REM*
REM*    calls : GSX (varptr(ptsout%), varptr(intout%), varptr(ptsin%),
REM*       varptr(intin%), varptr(contrl%))
REM*
REM*    to build   :      GSX-86 GSX-80
REM*                   CB86 DEMOCB CB80 DEMOCB
REM*    LINK86 DEMOCB,GSXCB86   LK80 DEMOCB,GSXCB80
REM*
REM*    history    :      Oct. 06, 1983 rhk
REM*
REM*************************************************************

DIM     CONTRL%(9), INTIN%(44), PTSIN%(71), INTOUT%(44), PTSOUT%(39)
DIM     PARAMADR%(4)
```

Listing 5-3. (continued)

```
DEF     GSX (ptsout.adr%, intout.adr%, ptsin.adr%,\
        intin.adr%, cntrl.adr%) EXTERNAL
rem     Setup for GDOS calls
FEND
DEF     bar (x1%, y1%, x2%, y2%)

REM     use General Drawing BAR Primitive

        contrl%(0) = 11
        contrl%(1) = 2
        contrl%(3) = 0
        contrl%(5) = 1
        ptsin%(0) = x1%
        ptsin%(1) = y1%
        ptsin%(2) = x2%
        ptsin%(3) = y2%
        CALL    GSX (paramadr%(4), paramadr%(3), paramadr%(2),\
                paramadr%(1), paramadr%(0))
        RETURN
FEND
DEF     set.attrib (fid%, inx%)
        contrl%(0) = fid%
        contrl%(1) = 0
        contrl%(2) = 1
        intin%(0) = inx%
        CALL    GSX (paramadr%(4), paramadr%(3), paramadr%(2),\
                paramadr%(1), paramadr%(0))
        RETURN
FEND
DEF     drawbar (x%, wide%)
           FOR i% = 1 TO 6
                x1% = x% : x2% = x% + wide% : y1% = x1% : y2% = x2%
   CALL set.attrib (24, i%)
                CALL bar(x1%, y1%, x2%, y2%)
                x% = x% + (wide% / 2)
           NEXT i%
           RETURN
FEND
DEF     doline (x1%, y1%, x2%, y2%)
        contrl%(0) = 6
        contrl%(1) = 2
        ptsin%(0) = x1%
        ptsin%(1) = y1%
        ptsin%(2) = x2%
        ptsin%(3) = y2%
        CALL    GSX (paramadr%(4), paramadr%(3), paramadr%(2),\
                paramadr%(1), paramadr%(0))
        RETURN
FEND
```

Listing 5-3. (continued)

```
DEF     drawfram (x1%, y1%, x2%, y2%)
        contrl%(0) = 6
        contrl%(1) = 5
        ptsin%(0) = x1% : ptsin%(6) = x1% : ptsin%(8) = x1%
        ptsin%(1) = y1% : ptsin%(3) = y1% : ptsin%(9) = y1%
        ptsin%(2) = x2% : ptsin%(4) = x2%
        ptsin%(5) = y2% : ptsin%(7) = y2%
        CALL    GSX (paramadr%(4), paramadr%(3), paramadr%(2),\
                paramadr%(1), paramadr%(0))
        RETURN
FEND
DEF     gclear
REM     Clear the Display surface
        contrl%(0) = 3
        contrl%(1) = 0
        CALL    GSX (paramadr%(4), paramadr%(3), paramadr%(2),\
                paramadr%(1), paramadr%(0))
        RETURN
FEND
DEF     gclose
        contrl%(0) = 2
        contrl%(1) = 0
        CALL    GSX (paramadr%(4), paramadr%(3), paramadr%(2),\
                paramadr%(1), paramadr%(0))
        RETURN
FEND
DEF     gopen
        contrl%(0) = 1
        contrl%(1) = 0
        contrl%(2) = 10
        intin%(0) = wid% : intin%(1) = 1 : intin%(2) = 1
        intin%(3) = 1 : intin%(4) = 1 : intin%(5) = 1
        intin%(6) = 1 : intin%(7) = 1 : intin%(8) = 1
        intin%(9) = 1
        CALL    GSX (paramadr%(4), paramadr%(3), paramadr%(2),\
                paramadr%(1), paramadr%(0))
        RETURN
FEND
DEF     ginit
        PRINT "Which output device -"
        PRINT "1 - CRT"
        PRINT "11 - PLOTTER"
        PRINT "21 - PRINTER"
        INPUT wid%
        PRINT "Open workstation £ ";wid%
        RETURN
FEND
```

Listing 5-3. (continued)

```
REM     Main Program
REM     Setup PARAMADR with the Addresses of the GSX Parameter Arrays

        PARAMADR%(0) = VARPTR(CONTRL%(0))
        PARAMADR%(1) = VARPTR(INTIN%(0))
        PARAMADR%(2) = VARPTR(PTSIN%(0))
        PARAMADR%(3) = VARPTR(INTOUT%(0))
        PARAMADR%(4) = VARPTR(PTSOUT%(0))

        xmin% = 0
        xmax% = 32767
        ymin% = 0
        ymax% = 32767

        CALL ginit
        CALL gopen
        CALL drawfram (xmin%, ymin%, xmax%, ymax%)
        CALL doline (xmin%, ymin%, xmax%, ymax%)
        CALL set.attrib (23, 2)
        CALL drawbar (1200, 1200)
        CALL set.attrib (23, 3)
        CALL drawbar (21968, 1200)
        i% = CONCHAR%
        CALL gclose

        STOP
END
```

End of Section 5

Section 6
FORTRAN-77

INTRODUCTION

This section tells you how to access GSX from FORTRAN-77. This section includes listings of the assembly language interface routine to GSX from Small and Large Memory Models from FORTRAN-77. A sample source listing follows the assembly language interface routines. Refer to the Digital Research FORTRAN-77 Language Reference Manual for more information about programming with FORTRAN-77.

The assembly language interface routine to GSX allows you to make graphics calls directly from FORTRAN-77 programs. The sample FORTRAN-77 program and its graphics output show how to use this interface module.

The interface routine sets the GDOS function code 0473 (HEX) in register CX, and the pointer to the parameter list arrays in DS:DX before it calls the disk operating system. Refer to the VDI specification in the GSX Graphics Extension Programmer's Guide for a detailed explanation of the parameter lists.

ARGUMENT PASSING

Contrl, intin, ptsin, intout, and ptsout are the five standard GSX parameter list array addresses. When your application program calls the GSX interface routine from FORTRAN-77, use the following calling sequence:

```
GSX (contrl, intin, ptsin, intout, ptsout)
```

Parameters from the FORTRAN-77 program are passed on the stack. Upon entry to the GSX assembly language interface routine, the top of the stack contains the return address to the example program, followed by the arguments stored in the same order as in the calling sequence. Refer to the Digital Research FORTRAN-77 Language Reference Manual for more information on stacks and how address space is allocated.

For example, upon entry to the GSX interface routine from a Small Memory Model FORTRAN-77 program, one word addresses are passed in the stack in the following manner:

```
STACK :     +0 Return Address
            +2 Address of Control Array
            +4 Address of Integer Input Array
            +6 Address of Point Input Array
            +8 Address of Integer Output Array
            +10 Address of Point Output Array
```

ARRAY DECLARATION

The GSX parameter list arrays can be declared in the FORTRAN-77 program as in the following example. The size of these parameter arrays depends on your application memory requirements.

```
int contrl[length_contrl] ;
int ptsin[length_ptsin] ;
int ptsout[length_ptsout] ;
int intin[length_intin] ;
int intout[length_intout] ;
```

INTERFACE ROUTINE

The following listing is for the FORTRAN-77, Small Memory Model.

Listing 6-1. FORTRAN-77 (Small Memory Model) to GSX-86 Assembly Language Interface Routine

```
;*****************************************************************
;                                                                 *
;  INTERFACE TO GSX-86 FROM FORTRAN-77                            *
;  (for Small Memory Model)                                       *
;  File name: GSXSMALL.A86                                        *
;                                                                 *
;  Calling sequence :                                             *
;          GSX (contrl, intin, ptsin, intout, ptsout)             *
;                                                                 *
;   Entry : return address on stack+0                             *
;          pointer to array contrl on stack+2                     *
;          pointer to array intin on stack+4                      *
;          pointer to array ptsin on stack+6                      *
;          pointer to array intout on stack+8                     *
;          pointer to array ptsout on stack+10                    *
;                                                                 *
;  Exit :   return address and parameters on stack unchanged      *
;                                                                 *
;  Notes : calls gdos with DS:DX points at the location of        *
;          parameter block on stack                               *
;                                                                 *
;                                                                 *
;  History : Sept. 28, 1983 rhk                                   *
;                                                                 *
;*****************************************************************
;
        cseg
        public    GSX

        GDOS    EQU  224
GSX:
        Push      bp                    ; Save caller BP.
        mov       bp, sp                ; Create new stack frame.
        Push      si                    ; Save caller SI.
        Push      di                    ; Save caller DI.
        Push      es                    ; Save caller ES.
```

Listing 6-1. (continued)

```
Setup GSX parameter block addresses on stack

Push    ss                      ; Points out array segment address
Push    word ptr 12[bp]           ; Points out array segment address
Push    ss                      ; Integer out array segment address
Push    word ptr 10[bp]           ; Integer out array offset address
Push    ss                      ; Points out array offset address
Push    word ptr 8[bp]            ; Points out array offset address
Push    ss                      ; Integer input array segment address
Push    word ptr 6[bp]            ; Integer input array offset address
Push    ss                      ; Control array segment address
Push    word ptr 4[bp]            ; Control array offset address

Mov     DX, SP                  ; DS:DX points to GSX parameter arrays
Mov     CX, 0473h                ; GSX Function number

INT     GDOS                    ; Invoke the GDOS

Add     sp, 20                    ; Clean up stack
Pop     es                      ; Restore caller ES.
Pop     di                      ; Restore caller DI.
Pop     si                      ; Restore caller SI.
Pop     bp                      ; Restore caller BP.
Ret                             ; Return

end
```

The following listing is the interface to GSX-86 from FORTRAN-77, Large Memory Model.

Listing 6-2. FORTRAN-77 (Large Memory Model) to GSX-86 Assembly Language Interface Routine

```
;*****************************************************************
;                                                                 *
;  INTERFACE TO GSX-86 FROM C and FORTRAN-77                      *
;  (Large Memory Model)                                           *
;                                                                 *
;  File name: GSXLARGE.A86                                        *
;                                                                 *
;  Calling sequence :                                             *
;          GSX (contrl, intin, ptsin, intout, ptsout)             *
;                                                                 *
;  Entry :   return address on stack+0                            *
;          pointer to array contrl on stack+4                     *
;          pointer to array intin on stack+8                      *
;          pointer to array ptsin on stack+12                     *
;          pointer to array intout on stack+16                    *
;          pointer to array ptsout on stack+20                    *
;                                                                 *
;  Exit :    return address and parameters on stack               *
;                                                                 *
;  Notes :   calls bdos with DS:DX points at the location of      *
;          parameter block address on stack                       *
;                                                                 *
;  History : Sept. 28, 1983 rhk                                   *
;                                                                 *
;  This routine points the data segment at the stack segment,     *
;  and DX at the GSX parameter block on the stack frame, returns  *
;  to the calling program without moving datas off the stack.     *
;                                                                 *
;*****************************************************************
;
        Cseg
        Public   GSX

        GDOS    EQU  224
GSX:
        Push     ds                ; Save caller DS.
        Push     bp                ; Save caller BP.
        Mov      bp, sp            ; Create new stack frame.
        Push     es                ; Save caller ES.
        Push     si                ; Save caller SI.
        Push     di                ; Save caller DI.

        Mov     ax, ss             ; Move stack segment into ax
        Mov     ds, ax             ; Set DS to SS
        Lea      dx, 8[bp]         ; DX points to GSX parameter array
                                   address
Mov     cx, 0473h ; GSX function number into cx
        Int GDOS        ; Call GDOS
```

Listing 6-2. (continued)

```
        Pop     di              ; Restore caller DI.
        Pop     si              ; Restore caller SI.
        Pop     es              ; Restore caller ES.
        Pop     bp              ; Restore caller BP.
        Pop     ds              ; Restore caller DS.
        Retf                    ; Return.

end
```

DIGITAL RESEARCH FORTRAN-77 EXAMPLE

The following listing is an example of a FORTRAN-77 source program that uses the assembly language interface routine for GSX.

Listing 6-3. Example FORTRAN-77 Source Listing

```
C*****************************************************************
C                                                                *
C  File name : DEMOF77.F77                                       *
C                                                                *
C  Purpose : Sample Fortran 77 program interfacing to GSX        *
C  using the assembly interface module, GSXSMALL.A86             *
C                                                                *
C  Calls : GSX (contrl(1), intin(1), ptsin(1),                   *
C  intout(1), ptsout(1))                                         *
C                                                                *
C  Uses : INCLUDE File "GSXDATA.F77"                             *
C                                                                *
C  To link : link86 demof77,gsxsmall (for small memory model)    *
C                                                                *
C  history : Oct. 04, 1983 rhk                                   *
C                                                                *
C*****************************************************************

   PROGRAM main

   C* EXAMPLE PROGRAM demonstrating interface to GSX from Fortran 77
   C* Link with assembly interface module, GSXSMALL.A86

        %INCLUDE "GSXDATA.F77"

        CHARACTER*22 signon
        CHARACTER*1 reply
```

Listing 6-3. (continued)

```
        signon = 'Demo Fortran 77 - GSX'
        xmin = 0
        ymin = 0
        xmax = 32767
        ymax = 32767

C  Open requested workstation
        CALL initgsx

C  Clear the display surface
        CALL clearit

C  Outline the display area
        CALL drawframe

C  Set line style to dotted line
        CALL setattrib(SETLINETYPE, DOTTEDLINE)

C  Draw a diagonal line
        CALL doline (xmin, ymin, xmax, ymax)

C  Set Character size
        CALL settxthgt (800)

C  Output text string
        CALL dotext (8192, 16384, signon, 22)

C  Set Fill pattern
        CALL setattrib (SETFILLSTYLE, 2)

C  Draw Bars
        CALL drawbar (1200, 1200)

C  Set Hatch fill style index
        CALL setattrib (SETFILLSTYLE, 3)

C  Draw more Bars
        CALL drawbar (21968, 1200)

C  Wait for a keyboard input
        READ (5,900) reply
900      FORMAT(A1)

C  Exit graphics
        CALL closeit
        STOP
        END

C  * End of Main Program ************************************
```

Listing 6-3. (continued)

```
C  ***** INITGSX ***********************************************
C  *     Get the graphics device id number
C  *     Open Graphics Device selected

        SUBROUTINE initgsx

        %INCLUDE 'GSXDATA.F77'

        INTEGER*2 device

        PRINT 'Which output device - '

        PRINT '1 - CRT'
        PRINT '11 - Plotter'
        PRINT '21 - Printer'
        PRINT '?'
        READ (5,100) device
100      FORMAT (I3)

C  ***** OPEN REQUESTED WORKSTATION*****************

        OPCODE = OPENWORKSTATION
        NPTSIN = ZERO
        LENINTIN = TEN

        WKID = device

C  *     Setup the initial default settings in integer input array
C  *     intin(1) - intin(10)

        LINETYPE = SOLIDLINE
        LINECOLOR = ONE
        MARKERTYPE = ONE
        MARKERCOLOR = ONE
        TEXTFONT = ONE
        TEXTCOLOR = ONE
        FILLSTYLE = HOLLOWFILL
        FILLINDEX = ONE
        FILLCOLOR = ONE

        CALL GSX (contrl(1), intin(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END
```

Listing 6-3. (continued)

```
C  ***** CLEAR WORKSTATION *************************
        SUBROUTINE clearit
        %INCLUDE 'GSXDATA.F77'
        OPCODE = CLEARWORKSTATION
        NPTSIN = ZERO
        CALL GSX (contrl(1), intin(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END

C  ***** DRAW A FRAME AROUND THE DISPLAY AREA ****

        SUBROUTINE drawframe
        %INCLUDE 'GSXDATA.F77'

C  Use polyline to draw the box
        OPCODE = POLYLINE
C  Five vertices passed in array points input - ptsin
        NPTSIN= 5
        ptsin(1) = xmin
        ptsin(2) = ymin
        ptsin(3) = xmax
        ptsin(4) = ymin
        ptsin(5) = xmax
        ptsin(6) = ymax
        ptsin(7) = xmin
        ptsin(8) = ymax
        ptsin(9) = xmin
        ptsin(10) = ymin
        CALL GSX (contrl(1), intin(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END

C  ***** SUPPORT OUTPUT PRIMITIVE ATTRIBUTE SETTINGS ****

        SUBROUTINE setattrib (cmd, inx)

        INTEGER*2 cmd, inx

        %INCLUDE 'GSXDATA.F77'

C  Set attribute index function
        OPCODE = cmd
        NPTSIN = ZERO
        LENINTIN = ONE
        intin(1) = inx
        CALL GSX (contrl(1), intin(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END
```

Listing 6-3. (continued)

```
C  ***** DRAWS a single segmented line using POLYLINE FUNCTION *
        SUBROUTINE doline (x1, y1, x2, y2)
        INTEGER*2 x1, y1, x2, y2
        %INCLUDE 'GSXDATA.F77'

        OPCODE = POLYLINE
        NPTSIN = 2
        ptsin(0) = x1
        ptsin(1) = y1
        ptsin(2) = x2
        ptsin(3) = y2
        CALL GSX (contrl(1), intin(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END

C  ***** SET CHARACTER HEIGHT **********************************
        SUBROUTINE settxthgt (height)
        INTEGER*2 height
        %INCLUDE 'GSXDATA.F77'

        OPCODE = SETCHARHEIGHT
        NPTSIN = ONE

C  Set points input X coordinate = 0
        C Y coordinate = height in device units
        ptsin(1) = ZERO
        ptsin(2) = height
        CALL GSX(contrl(1), intin(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END

C  ***** OUTPUT TEXT ********************************************

        SUBROUTINE dotext (x, y, string, len)

        INTEGER*2 x, y, len, string(80)

        %INCLUDE 'GSXDATA.F77'

        OPCODE = TEXT
        NPTSIN = ONE
        NCHAR = len
        TEXTXPOS = x
        TEXTYPOS = y
        CALL GSX (contrl(1), string(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END
```

Listing 6-3. (continued)

```
C  ***** DRAW 6 BARS USING GDP BAR FUNCTION ***********

        SUBROUTINE drawbar (x, wide)

        INTEGER*2 x, wide
        INTEGER*2 left, bottom, right, top

        %INCLUDE 'GSXDATA.F77'

        DO 1000 i = 1, 6
C       draw six Bars
        left = x
        bottom = x
        right = x + wide
        top = right
        CALL setattrib(SETSTYLEINDEX, i)
        CALL bar(left, bottom, right,top)
        x = x + (wide / 2)
1000     CONTINUE

        RETURN
        END

        SUBROUTINE bar (left, bottom, right, top)

        INTEGER*2 left, bottom, right, top

        %INCLUDE 'GSXDATA.F77'

C  Use GDP BAR function

        OPCODE = DRAWINGPRIM
        LENINTIN = ZERO
        NPTSIN = TWO
        GDPID = ONE

C  Lower left and upper right vertices passed in ptsin array
        ptsin(1) = left
        ptsin(2) = bottom
        ptsin(3) = right
        ptsin(4) = top
        CALL GSX (contrl(1), intin(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END
```

Listing 6-3. (continued)

```
C  ***** CLOSE WORSTATION **********************

        SUBROUTINE closeit

        %INCLUDE 'GSXDATA.F77'

        OPCODE = CLOSEWORKSTATION
        NPTSIN = ZERO
        CALL GSX (contrl(1), intin(1), ptsin(1), intout(1), ptsout(1))
        RETURN
        END
```

The following is the file GSXDATA.F77 for the DEMOF77.F77 file.

```
C  * Declare the five GSX parameter list arrays

        INTEGER*2 contrl(20)
        INTEGER*2 intin(100)
        INTEGER*2 ptsin(100)
        INTEGER*2 intout(100)
        INTEGER*2 ptsout(100)
        INTEGER*2 xmin, ymin, xmax, ymax

C  * common GSX PARAMETER array definitions

        COMMON /GSXARRAY/ contrl, intin, ptsin, intout, ptsout
        COMMON /EXTENT/ xmin, xmax, ymin, ymax

C  Define the parameters for GSX opcodes

        PARAMETER (OPENWORKSTATION = 1)
        PARAMETER (CLOSEWORKSTATION = 2)
        PARAMETER (CLEARWORKSTATION = 3)
        PARAMETER (POLYLINE = 6)
        PARAMETER (TEXT = 8)
        PARAMETER (FILLAREA = 9)
        PARAMETER (DRAWINGPRIM = 11)
        PARAMETER (SETCHARHEIGHT = 12)
        PARAMETER (SETLINETYPE = 15)
        PARAMETER (SETFILLSTYLE = 23)
        PARAMETER (SETSTYLEINDEX = 24)
```

Listing 6-3. (continued)

```
      PARAMETER (GDPBAR = 1)
      PARAMETER (SOLIDLINE = 1)
      PARAMETER (DASHEDLINE = 2)
      PARAMETER (DOTTEDLINE = 3)
      PARAMETER (DASHEDDOTTED = 4)
      PARAMETER (HOLLOWFILL = 0)
      PARAMETER (SOLIDFILL = 1)
      PARAMETER (PATTERNFILL = 2)
      PARAMETER (HATCHFILL = 3)

      PARAMETER (YES = 1)
      PARAMETER (NO = 0)

      PARAMETER (ZERO = 0)
      PARAMETER (ONE = 1)
      PARAMETER (TWO = 2)
      PARAMETER (THREE = 3)
      PARAMETER (FOUR = 4)
      PARAMETER (TEN = 10)

C  Define the integer input array entries for open workstation defaults

      INTEGER*2 WKID, LINETYPE, LINECOLOR
      INTEGER*2 MARKERTYPE, MARKERCOLOR
      INTEGER*2 TEXTFONT, TEXTCOLOR
      INTEGER*2 FILLSTYLE, FILLINDEX, FILLCOLOR

C  Define equates to the control array entries

      INTEGER*2 OPCODE, NPTSIN, NPTSOUT, NCHAR
      INTEGER*2 LENINTIN, LENINTOUT, GDPI
      INTEGER*2 TEXTXPOS, TEXTYPOS

      EQUIVALENCE (WKID, intin(1))
      EQUIVALENCE (LINETYPE, intin(2))
      EQUIVALENCE (LINECOLOR, intin(3))
      EQUIVALENCE (MARKERTYPE, intin(4))
      EQUIVALENCE (MARKERCOLOR, intin(5))
      EQUIVALENCE (TEXTFONT, intin(6))
      EQUIVALENCE (TEXTCOLOR, intin(7))
      EQUIVALENCE (FILLSTYLE, intin(8))
      EQUIVALENCE (FILLINDEX, intin(9))
      EQUIVALENCE (FILLCOLOR, intin(10))
```

Listing 6-3. (continued)

```
C*
      EQUIVALENCE (OPCODE,contrl(1))
      EQUIVALENCE (NPTSIN,contrl(2))
      EQUIVALENCE (NPTSOUT,contrl(3))
      EQUIVALENCE (LENINTIN,contrl(4))
      EQUIVALENCE (LENINTOUT,contrl(5))
      EQUIVALENCE (GDPID, contrl(6))
      EQUIVALENCE (NCHAR, contrl(7))
      EQUIVALENCE (TEXTXPOS, ptsin(1))
      EQUIVALENCE (TEXTYPOS, ptsin(2))
```

End of Section 6

Index

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/