Professional Graphics Series

IBM RT PC Graphics Development Toolkit

Programming Family

Personal
Computer
Software

59X8554

LIMITATIONS OF REMEDIES

IBM's entire liability and your exclusive remedy shall be as follows:

1. With respect to defective media during the warranty period:
   a. IBM will replace media not meeting IBM's "Limited Warranty" which is returned to IBM or an IBM authorized representative with a copy of your receipt.
   b. In the alternative, if IBM or such IBM authorized representative is unable to deliver replacement media which is free of defects in materials and workmanship, you may terminate this Agreement by returning the program and your money will be refunded.
2. With respect to warranted programs, in all situations involving performance or nonperformance during the warranty period, your remedy is (a) the correction by IBM of program defects, or (b) if, after repeated efforts, IBM is unable to make the program operate as warranted, you shall be entitled to a refund of the money paid or to recover actual damages to the limits set forth in this section.

   For any other claim concerning performance or nonperformance by IBM pursuant to, or in any other way related to, the warranted programs under this Agreement, you shall be entitled to recover actual damages to the limits set forth in this section.

IBM's liability to you for actual damages for any cause whatsoever, and regardless of the form of action, shall be limited to the greater of $5,000 or the money paid for the program that caused the damages or that is the subject matter of, or is directly related to, the cause of action.

In no event will IBM be liable to you for any lost profits, lost savings or other incidental or consequential damages arising out of the use of or inability to use such program even if IBM or an IBM authorized representative has been advised of the possibility of such damages, or for any claim by any other party.

SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

SERVICE

Service from IBM, if any, will be described in Program Specifications or in the statement of service, supplied with the program, if there are no Program Specifications.

IBM may also offer separate services under separate agreement for a fee.

GENERAL

Any attempt to sublicense, assign, rent or lease, or, except as expressly provided for in this Agreement, to transfer any of the rights, duties or obligations hereunder is void.

This Agreement will be construed under the Uniform Commercial Code of the State of New York.

Program Specification

IBM RT Personal Computer Graphics Development Toolkit Licensed Program (55X8921)

Statement of Limited Warranty

The IBM RT Personal Computer[1] Graphics Development Toolkit Licensed Program is warranted to conform to this Program Specification when properly used in its designated operating environments.

Any other documentation with respect to this licensed program is provided for information purposes only and does not extend or modify this IBM RT PC Graphics Development Toolkit Licensed Program Program Specification.

The IBM RT PC Graphics Development Toolkit Licensed Program Program Specification may be updated from time to time. Such updates may constitute a change to these specifications.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

This limited warranty and the 90-day program media warranty are contained in the IBM Program License Agreement supplied with this product. These warranties are available to all licensees of the IBM RT PC Graphics Development Toolkit Licensed Program. The limited warranty period is until February 1, 1988, or until six months after written notice by IBM that the warranty period has been terminated, whichever is sooner.

Statement of Function Warranted

The IBM RT Personal Computer Graphics Development Toolkit Licensed Program (55X8921) provides tools for programmers who develop graphics applications. The Toolkit includes a set of graphics primitives that can be called by high level languages to perform functions such as displaying pie slices and drawing lines, polygons, and circles, using the Virtual Device Interface (VDI).

The highlights of this licensed program are:

- Supports the IBM RT PC Virtual Device Interface (VDI) between devices and programs
- Helps provide device independence for programs
- Provides graphic and text functions via graphic subroutines that include:
  - Circle, arc, pie slice, and bar charts
  - Multiple colors and fill patterns
  - Polyline, polymarker
  - Multiple font cursor text
  - Rotatable , scalable, multiple font graphic text
  - Raster operations
  - Input operations.
- Provides a program interface via language bindings to IBM RT PC FORTRAN 77, IBM RT PC BASIC Interpreter and Compiler (compiled only), IBM RT PC Pascal, and the C compiler

[1] RT, RT PC, and RT Personal Computer are trademarks of IBM

provided with the IBM RT PC AIX[2] Operating System.

Specified Operating Environment

Machine Requirements

The minimum machine requirements are:

- An IBM RT PC with a display (for example, the IBM RT PC Advanced Color Graphics Display, IBM RT PC Advanced Monochrome Graphics Display, the IBM Personal Computer Display, or an equivalent display).

Note: The number of users on the IBM RT PC AIX Operating System Licensed Program, Version 1.1., (74X9995), the number and type of tasks, and the application requirements may expand the requirements beyond these minimums.

Programming Requirements

The IBM RT PC AIX Operating System Licensed Program, Version 1.1., (74X9995) is a prerequisite for program execution.

Statement of Service

Program service for valid program-related defects in the IBM RT PC Graphics Development Toolkit Licensed Program is available to all IBM RT PC Graphics Development Toolkit Licensed Program licensees until February 1, 1988, or until six months after written notice by IBM that the warranty period has been terminated, whichever is sooner. However, service will be provided only for the current update level and for the prior release for ninety (90) days following release of the current level of update.

Each licensee's access to program service is determined by the marketing channel through which the license was obtained. For example, in the United States and Puerto Rico, if the IBM RT PC Graphics Development Toolkit Licensed Program license was obtained through:

- An authorized IBM Personal Computer dealer.

  Requests for program service should be made through your dealer.

- The IBM North-Central Marketing Division or the IBM South-West Marketing Division.

  Your company will have established a technical support location to interface to IBM central service through an IBM Support Center, and your request for program service should be made through your company's technical support location.

If the IBM RT PC Graphics Development Toolkit Licensed Program is obtained through transfer of license from another party under the conditions of the IBM Program License Agreement supplied with this product, the new licensee may obtain program service through the access arrangement provided for the original licensee.

When a license is transferred, if the original license was obtained through the IBM North-Central Marketing Division or the IBM South-West Marketing Division, the old licensee is responsible for contacting their IBM marketing representative to make arrangements to transfer service entitlement to the new licensee; the new licensee must also establish a qualified technical support location to interface to IBM central service.

IBM does not guarantee service results or that the program will be error free, or that all program defects will be corrected.

IBM will respond to a reported defect in an unaltered portion of a supported release of the licensed program by issuing: defect correction information such as correction documentation, corrected code, or notice of availability of corrected code; a restriction; or a bypass.

---

2 AIX is a trademark of IBM

Corrected code is provided on a cumulative basis on diskettes; no source code is provided. Only one copy of the corrections with supporting documentation will be issued to the licensee, or the agent of the licensee, reporting the defect. IBM will authorize various agents such as the IBM Personal Computer dealers and the IBM North-Central Marketing Division or IBM South-West Marketing Division customer's technical support locations to make and distribute a copy of the corrections if needed, to each IBM RT PC Graphics Development Toolkit Licensed Program licensee which they serve.

IBM will notify authorized IBM Personal Computer dealers, IBM marketing and service representatives, and IBM North-Central Marketing Division and IBM South-West Marketing Division customer's technical support locations if and when an update is made available. Program updates contain all currently available changes for the licensed program.

Licensees may request available updates to this licensed program, if any, prior to the program service termination date. As with defect corrections, IBM will authorize various agents such as IBM Personal Computer dealers and the IBM North-Central Marketing Division and IBM South-West Marketing Division customer's technical support locations to distribute a copy of the update, if needed, to each IBM RT PC Graphics Development Toolkit Licensed Program licensee which they serve.

The total number of copies of an update distributed to IBM RT PC Graphics Development Toolkit Licensed Program licensees within a customer's location may not exceed the number of copies of the IBM RT PC Graphics Development Toolkit Licensed Program licensed to the customer.

IBM does not plan to release updates of IBM RT PC Graphics Development Toolkit Licensed Program code on a routine basis for preventative service purposes. However, should IBM determine that there is a general need for a preventative service update, it will be made available to all licensees through the same process that is utilized to distribute general IBM RT PC Graphics Development Toolkit Licensed Program updates, as described above.

Following the discontinuance of all program services, this program will be distributed on an "As Is" basis without warranty of any kind either express or implied.

June 1986 Printed in U.S.A.
(GC23-0955-1)

IBM Corporation
Industry Systems Products
472 Wheelers Farms Road
Milford, Connecticut 06460

Professional Graphics Series

IBM RT PC Graphics Development Toolkit

Programming Family

IBM

Personal
Computer
Software

First Edition (July 1986)

Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

About This Book

This book explains how to use the IBM RT Personal Computer Graphics Development Toolkit, a graphics software package that supports the Virtual Device Interface (VDI). The intent of this book is to provide you with an overview of Toolkit operation and with specific information about how to incorporate Toolkit routines into your graphics application program.

Who Should Read This Book

This book is written for people who write graphics application programs. To use this product, you should be familiar with the IBM RT PC AIX[1] Operating System and with the general use of graphics software. In addition, you should know how to program in one or more of the following languages:

- Pascal
- C
- BASIC
- FORTRAN.

[1] AIX is a trademark of IBM.

Before You Begin

Before you begin using Graphics Development Toolkit, ensure that you have the minimum requirements for both hardware and software.

Hardware Requirements

The minimum hardware required to use Graphics Development Toolkit is one of the supported graphics output devices such as a display, printer, or plotter. For a list of the supported devices, see the table at the beginning of Appendix C, "Graphics Drivers", in this manual.

Software Requirements

The IBM RT PC Graphics Development Toolkit device drivers are provided as part of the IBM RT PC Multi-User programs.

The IBM RT PC Graphics Development Toolkit provides language bindings for the BASIC, C, Pascal, and FORTRAN languages. The IBM RT PC C Compiler is included with the IBM RT PC AIX Operating System. In order to use BASIC, Pascal, or FORTRAN, you must have the appropriate language compiler, as follows:

- IBM RT PC BASIC Interpreter and Compiler
- IBM RT PC Pascal
- IBM RT PC FORTRAN 77.

How to Use this Book

The first two chapters are intended as an overview, to help you understand the basic operation of the Graphics Development Toolkit. It is recommended that you read these chapters first.

Chapter 3 is intended as a reference section, to obtain an understanding of the purpose and operation of each individual routine. Use this chapter, and the appropriate language reference booklet, to incorporate the Toolkit routines into an application program.

Organization

This book is organized into three chapters, four appendixes, a glossary, and an index, as follows:

- **Chapter 1,** "Introduction", describes this product and its advantages. A description of Normalized Device Coordinates (NDC) and text capabilities is also presented.
- **Chapter 2,** "Programming Considerations", contains information about NDCs, aspect ratios, and helpful hints for programming applications.
- **Chapter 3,** "Toolkit Routines", describes the purpose and operation of each routine. The routines are organized into the following functional groups:

  – Workstation Control Routines
  – Paging Routines
  – Pel Routines
  – Cursor Control Routines
  – General Graphics Routines
  – Graphics Primitives
  – Graphics Text Routines
  – Alpha Text Routines
  – Input Routines
  – Error Handling.

For the correct syntax of each routine, refer to the language reference booklets that accompany this manual.

- **Appendix A,** "Installing the Graphics Development Toolkit", contains information about installing the software for the Graphics Development Toolkit and establishing the proper operating parameters within the system environment.
- **Appendix B,** "Example Programs", contains sample programs and the corresponding output created by these programs.
- **Appendix C,** "Graphics Drivers", contains information about capabilities and limitations of specific graphic peripheral devices.
- **Appendix D,** "Error Codes", lists and describes the error codes that can be returned by the Toolkit.

- The **Glossary** provides a definition of many of the technical terms that appear in the documentation for the Graphics Development Toolkit.
- The **Index** shows where to find information about several of the topics that are discussed within this manual.

Note: Language reference booklets are provided for each of the languages supported by the Graphics Development Toolkit. Each booklet is a quick reference to the syntax needed to include each routine in an application program.

Highlighting

Throughout this manual, bold italics have been used to highlight the first occurrence of new terms. In the descriptions of the Toolkit routines, parameter names are italicized. Computer output and user input are shown in monospace type and in color.

Related Information

The following books contain related information that you might find useful.

IBM RT PC Plotting System Programmer's Guide
This book provides information about installation procedures, plotting concepts, and Plotting System routines. Included with the Plotting System guide are language reference booklets that support the C, BASIC, FORTRAN, and Pascal programming languages.

IBM RT PC Graphical File System Programmer's/User's Guide
This book is a guide to the installation and use of the Graphical File System. It contains information about Metafile concepts and elements, interactive interpretation of metafiles and programming with the Graphical File System.

IBM RT PC Graphics Terminal Emulator User's Guide
This book is a guide to the installation and use of the Graphics Terminal Emulator. This book contains information about defining communications protocol to allow communications with a host computer, transferring data to and from a host computer, and redirecting graphics output from the host computer to a plotter, printer, or other output device.

IBM RT PC C Language Guide and Reference
This book provides information for writing, compiling, and running C language programs.

IBM RT PC BASIC Language Reference
This book is an encyclopedia-type manual. It contains syntax and format for BASIC language commands, statements, and functions, arranged in alphabetical order.

IBM RT PC BASIC Language Handbook
This book contains general information about using BASIC. Some sections of the book provide help for those unfamiliar with BASIC. Other sections contain information on advanced subjects for the experienced programmer.

IBM RT PC Pascal Compiler Fundamentals
This book presents the fundamentals of the IBM RT PC Pascal Compiler Version 1.00.

IBM RT PC Pascal Compiler Language Reference
This book is designed as a reference tool and contains information about specific keywords, library routines, and metacommands used in writing programs.

IBM RT PC FORTRAN 77
This book is designed as a reference tool and discusses the implementation of FORTRAN 77.

IBM RT PC Using and Managing the AIX Operating System
This manual contains information for using the IBM RT PC AIX Operating System commands, working with the file system, developing shell procedures, and performing such system management tasks as creating and mounting file systems, backing up the system and repairing file system damage.

IBM RT PC AIX Operating System Commands Reference
This manual describes the IBM RT PC AIX Operating System commands, including the following:

- The proper syntax for each command with the acceptable flags and arguments
- Examples showing proper usage of the commands.

IBM RT PC Messages Reference
This manual lists the errors you may see on your display, and shows how to respond to the messages.

IBM RT PC AIX Operating System Technical Reference
This manual gives details about the IBM RT PC AIX Operating System, the file system, files, special files, miscellaneous files, and writing device drivers.

IBM RT PC Installing and Customizing the AIX Operating System
This guide explains how to install the programs contained in the IBM RT PC AIX Operating System Licensed Program Product, as well as how to install other licensed program products. This guide also explains how to customize the AIX Operating System.

IBM RT PC User Setup Guide
This guide contains information about identifying, unpacking, and connecting devices and testing the system setup.

Ordering Additional Copies of This Book

To order additional copies of this publication (without the program diskettes), use either of the following sources:

- To order from your IBM representative, use Order Number SV21-8058.
- To order from your IBM dealer, use Part Number 55X8922.

A binder and language reference booklets for C, BASIC, FORTRAN, and Pascal are included with the order.

About the IBM RT PC Professional Graphics Series

The IBM RT PC Professional Graphics Series consists of several compatible programs and libraries that provide you with a comprehensive graphics package. Designed to run under the IBM RT PC AIX Operating System, the series provides programmers with a versatile way to create, modify, store, and output high-quality graphic images.

Using the series, you can develop custom graphics applications that meet the needs of the business, scientific, engineering, and research environments.

The Series Components

The IBM RT PC Professional Graphics series includes:

- IBM RT PC Graphics Development Toolkit
- IBM RT PC Plotting System
- IBM RT PC Graphical File System
- IBM RT PC Graphics Terminal Emulator.

The following pages briefly describe the series components. See the individual manuals for more information.

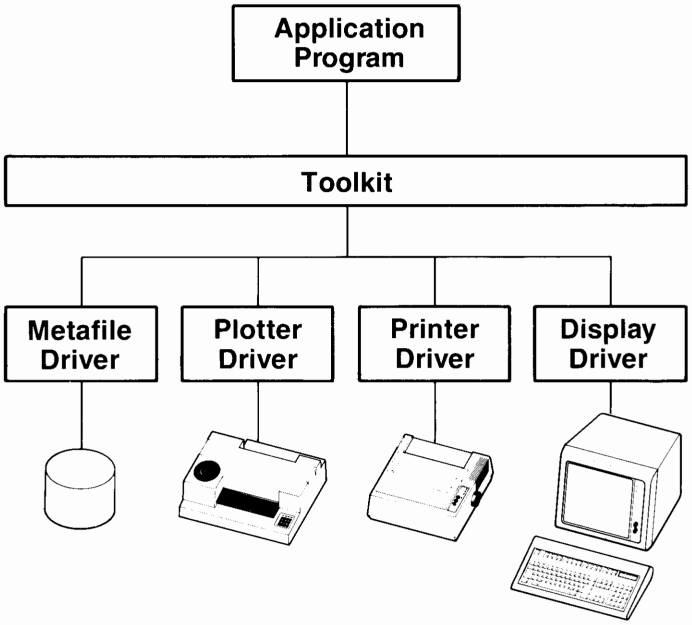
IBM RT PC Graphics Development Toolkit

At the heart of the Professional Graphics Series is the Graphics Development Toolkit, including its device drivers. The device drivers, part of the IBM RT PC Multi-User programs, must be installed in order to use any of the other graphics products in the series. The Toolkit controls the exchange of information (commands and data) between the high-level, device-independent applications and the device-dependent drivers. These drivers, in turn, control the operation of a variety of input and output devices, such as printers, plotters, and displays.

The Toolkit converts coordinates from a writing surface to the viewing surface coordinate systems of individual graphics devices. In addition, it manages all supported devices connected to the system and fulfills program requests for device information.

A library of language *bindings* (interfaces) is also contained in the Graphics Development Toolkit. Using this product, you can develop application programs that access the Toolkit directly. Language Reference Booklets for C, Pascal, FORTRAN 77, and BASIC are provided with the software.

The following illustration shows the interface between an application program and the Graphics Development Toolkit.



IBM RT PC Plotting System

Plotting System consists of subroutines that allow you to create professional-quality charts. The types of charts available include area, bar, line, pie, scatter, schedule, step and text-only charts.

Plotting System provides bindings to several major languages. The Toolkit, although transparent to the programmer, is the foundation of Plotting System.

Because Plotting System provides a comprehensive set of default values, you can create charts with a minimum number of steps. A built-in chart dimensioning procedure ensures that the sizing and spacing of chart components are consistent and proportional. Multiple charts or chart types can appear on a single display surface.

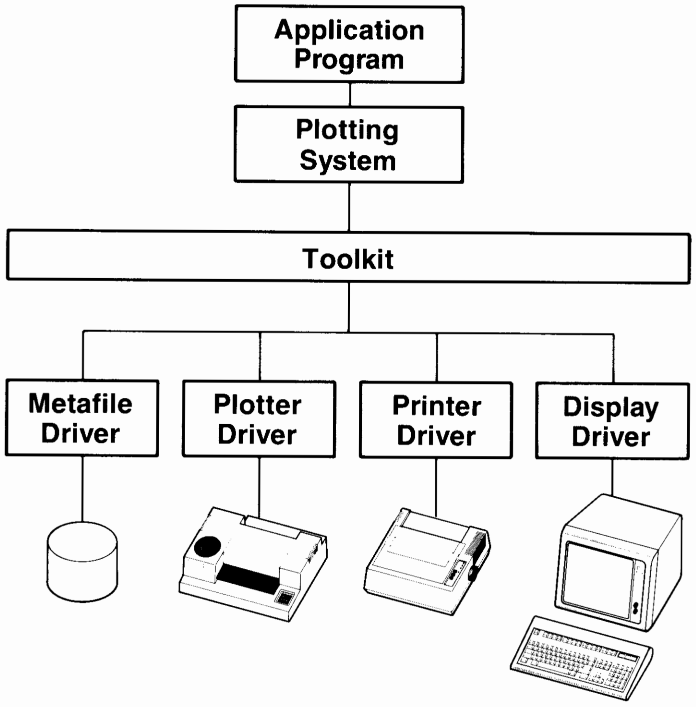
Most chart attributes can be changed. These include:

- Line style for line and step charts
- Fill pattern for area, pie and bar charts

- Color
- Text height and font
- Logarithmic axes
- Axis range
- Tick-spacing.

Plotting System includes facilities for both graphic and text annotation. In addition, it contains device inquiry capabilities that allow you to use device-independent features. You can create interactive graphics applications by using input capabilities, such as choice, locator and string.

The following illustration shows an application program interface with Plotting System.



When programming with Plotting System, include calls to Plotting System routines in your application program. The external references are resolved when the compiled program is linked to the subroutine library.

To use Plotting System, you should be an experienced programmer familiar with the BASIC, C, FORTRAN or Pascal languages.

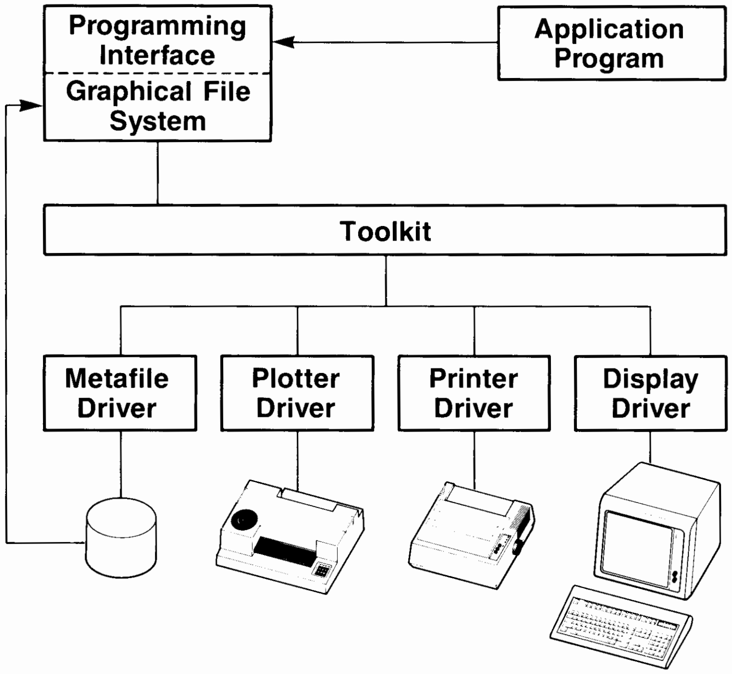
IBM RT PC Graphical File System

The Graphical File System allows you to retrieve and interpret *metafiles* that have been generated by the Metafile Device Driver within the Graphics Development Toolkit. Each metafile contains a series of pictures or graphics images that have been stored in a special metafile format. This format promotes standardization and portability between different kinds of graphics output devices.

You can use the Graphical File System as a *programming interface* or an *interactive interface.*

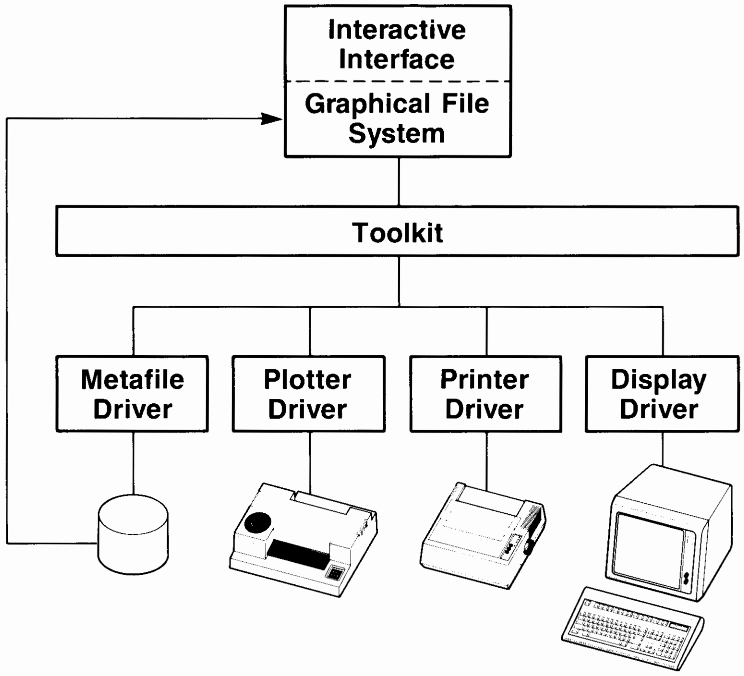
The programming interface consists of subroutines called by an application program. These subroutines are in the form of functions that retrieve metafile pictures and interpret individual *elements* within a picture. The programming interface requires programming experience in BASIC, C, FORTRAN or Pascal.

A diagram of the programming interface is shown in the following illustration.



The interactive interface, on the other hand, is an easy-to-use, icon-driven program. By selecting icons, you can perform metafile interpretation functions. To operate the interactive interface, you need no special knowledge of either graphics or programming.

The interactive interface is diagrammed in the following illustration.



IBM RT PC Graphics Terminal Emulator

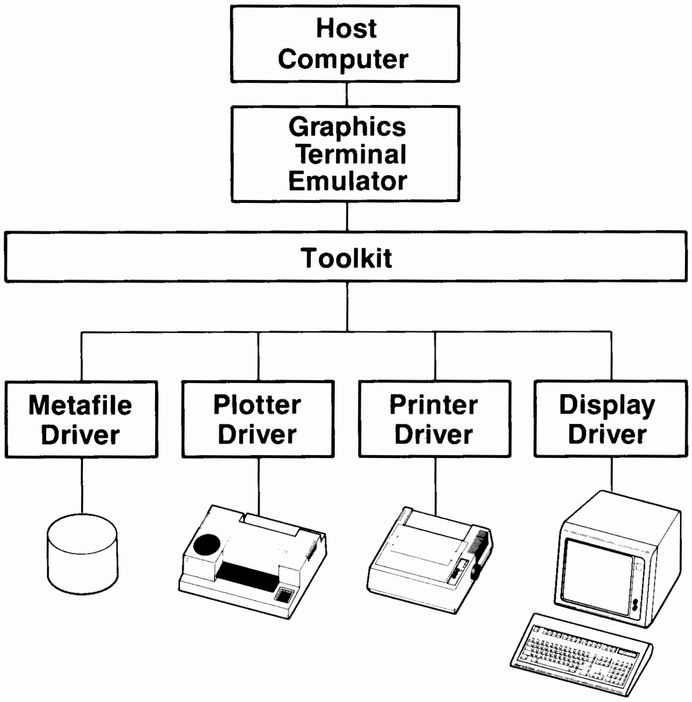
Graphics Terminal Emulator allows you to emulate most functions of the Tektronix[2] 4014 graphics terminals, most functions of the Tektronix 4014 with the Extended Graphics Module, and selected features of the Tektronix 4105. Most functions of the Lear Siegler ADM[3]-3A are also supported.

Graphics Terminal Emulator allows you to access host-computer software, send and receive files, and print both alphanumeric and graphic output. In addition, it reproduces the effects of the many special function keys available on the Tektronix family of supported terminals.

Because Graphics Terminal Emulator is an icon-driven system, no special knowledge of programming is required to use it. Experience on the supported terminals, however, is necessary to create graphics.

[2] Tektronix is a trademark of Tektronix, Inc.
[3] ADM is a trademark of Lear Siegler, Inc.

The operation of Graphics Terminal Emulator involves an external host computer, as shown in the following illustration.



Contents

Figures

Chapter 1. Introduction

CONTENTS

About this Chapter

This chapter contains an overview of the Graphics Development Toolkit. It includes information about the Virtual Device Interface (VDI) and the following Toolkit concepts:

- Drawing Primitives
- Workstations
- Graphics Coordinates
- Aspect Ratio
- Device Drivers
- The Toolkit Routines.

The Toolkit

The Graphics Development Toolkit provides the means for writing device-independent graphics software. *Device-independent* means you can direct your application program output to any supported workstation (an input and/or output graphics device) without having to modify the application.

With the Toolkit, graphics can be made an integral part of an accounting program, a word processor, a building layout, or an educational program. Since device drivers now exist for a large number of workstations, system configuration tasks become a smaller part of the programmer's concern.

The Toolkit consists of:

- A set of device drivers (for displays, printers, plotters, and metafiles)
- A library of routines that perform various graphics and text functions
- Language reference booklets containing the specific language syntax for each routine.

Virtual Device Interface

The Virtual Device Interface (VDI) is based upon the concept of a virtual device; a device without physical form that can be used as a reference model during the development of both device drivers and application programs. With the VDI, application programs can be written independent of specific hardware devices. Only the device driver need be specific to the selected device.

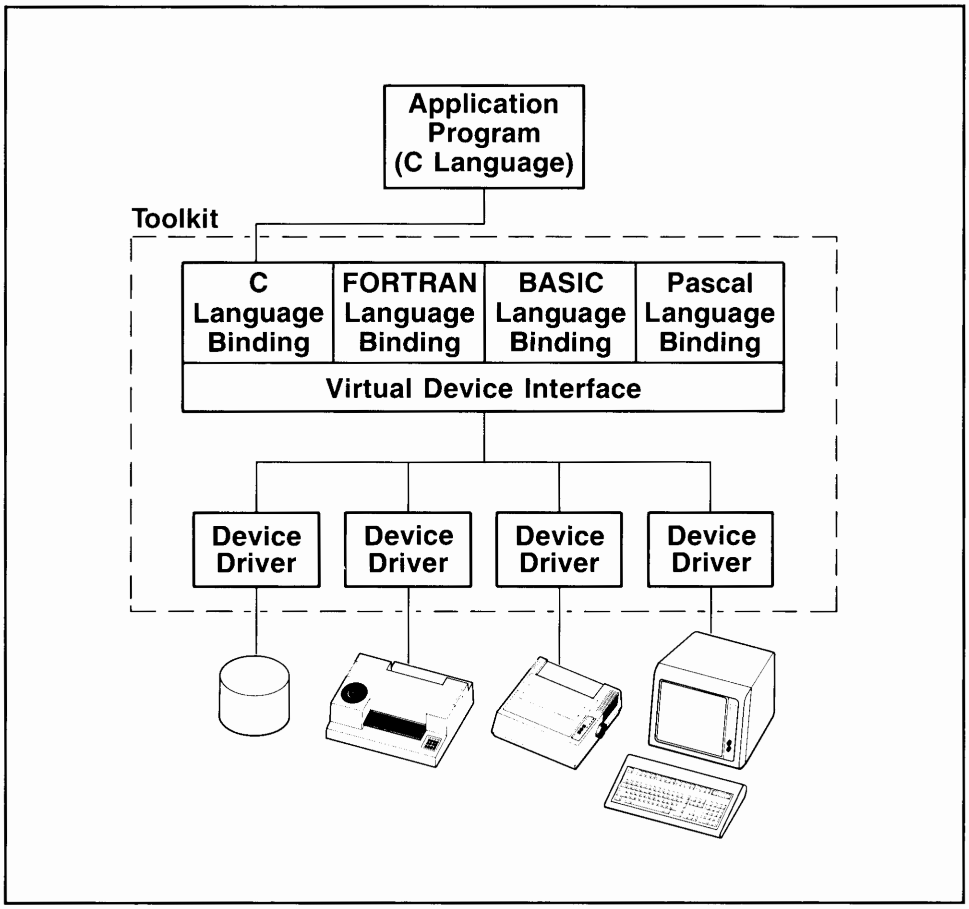


Figure 1-1. VDI Software Overview

The VDI defines a common language (protocol) which allows device-independent software and the device-dependent drivers to communicate. This protocol consists of predefined functional capabilities, accessing methods, and parameter-passing conventions that enable the software to produce the required results.

The Graphics Development Toolkit contains a list of graphics and text routines associated with the VDI. The implementation of these routines makes all workstations appear as "identical" virtual devices.

Drawing Primitives

Drawing primitives include polylines, polymarkers, and text. The polyline primitive draws vectors between sequences of end points, which are specified as an array. The polymarker is similar to the polyline, except it places a marker symbol (such as a star, asterisk, or square) at each point in the array. The text primitive displays text strings at any position with any orientation.

Three types of text (alpha, graphics, and cursor text) facilitate the integration of text and graphics, as well as design-efficient menu-oriented interfaces (see Chapter 3, "Toolkit Routines", for the specifics).

The VDI also supports raster devices, fill and cell array primitives, and raster operations (known as Pel routines). The fill operation paints the interior of a polygon with a specified color or pattern. The cell array primitive allows a two-dimensional array of pels (pixels) of different colors to be defined. Cell replication over a specified area is accomplished by specifying desired boundary conditions.

The Pel routines move one or more pels by either copying them from one location on the display screen to another or storing them for later use (see Chapter 3, "Toolkit Routines", for the specifics).

Special Workstation Capabilities

Some workstations have special capabilities, such as the ability to draw circles, arcs, or bars. The VDI provides access to these capabilities through a special graphics mechanism called the Generalized Drawing Primitive (GDP).

The VDI manages the attributes associated with each output primitive. For example, polylines have line type (such as solid, dash, or dotted), width, and color attributes. Polymarkers are associated with attributes of type, size, and color. Text primitive attributes include size, color, and orientation. In addition, fonts (multiple character sets) can be accessed if they are available in the workstation being addressed (see Chapter 3, "Toolkit Routines", for the specifics).

Most workstations cannot perform every graphics function. However, device drivers, in conjunction with the VDI, can be made to emulate additional functions, as part of their task of making all workstations appear the same. To aid programmers, the VDI also provides for inquiry operations. The inquiry operations allow the application programmer to determine workstation capabilities as well as primitive attribute status and viewing operations.

VDI Input Operations

The VDI supports a set of input operations that present several different types of information to the application program. These operations include Input Locator, Input Valuator, Input Choice, Input String, and Read Cursor Keys.

- **Input Locator** enables the user to position a graphics cursor at a particular point on the workstation, and return that location to the application program.
- **Input Valuator** indicates the current value of a continuous input device such as those that are potentiometer-based.
- **Input Choice** returns an integer value that represents one of a set of choices. An example might be one of the function keys. See Chapter 3, "Toolkit Routines", for more specific information.
- **Input String** accepts a string of characters from the keyboard.
- **Read Cursor Keys** determines whether a cursor movement key was struck and returns the resulting direction in integer form.

Graphics Devices

The VDI allows graphics and alphanumeric application programs to operate with IBM graphics devices such as printers, plotters, and displays.

Graphics Coordinates

The VDI can produce similar kinds of graphics output on workstations of different sizes and shapes. This is done by converting a Normalized Device Coordinate (NDC) associated with the VDI into a Device Coordinate (DC) that is specific to the selected workstation.

There are two ways that NDC conversion can take place. In the first way, NDC units range from 0-32767 and are mapped (transformed) to the full extent of the physical workstation surface on each axis. In the second way, NDC units are mapped to equal physical distances on both axes. For more information on NDC units, refer to Chapter 2, "Programming Considerations."

Aspect Ratio

The aspect ratio is a ratio of the horizontal and vertical dimensions of an image. The ability to maintain or control this ratio is important in the transfer and reproduction of an image on various types of display screens or hardcopy devices. This is done by converting Normalized Device Coordinates (NDC) used by an application into actual device coordinates for each workstation. If the actual device coordinates do not match the NDC coordinates, aspect ratio must be taken into account.

Output can maintain its originally designed proportions or take advantage of the full device coordinates. As a result, circles will always be circles. For more information on aspect ratio, refer to Chapter 2, "Programming Considerations."

Device Drivers

Device drivers communicate directly with a graphics device. These drivers are installed when the Toolkit software is installed. However, certain environmental parameters must be set in order for them to be used. Once installed, and configured properly, the drivers are referenced by the application program, allowing your application to run like any other program.

Each workstation is controlled by a device driver. For your application program to use a variety of workstations, each device driver must translate the information that passes between its workstation and the application program. The device driver will be different for each unique workstation, because the translation process is device-specific.

Graphics/Cursor Mode

The Enter Cursor Addressing Mode and Exit Cursor Addressing Mode routines allow you to select between the all-points-addressable (APA) graphics mode and the alphanumeric (A/N) cursor mode. You select the mode best suited for each portion of your application. The default is the graphics mode.

Graphics Mode

A workstation opens in the graphics mode. To enter the graphics mode when in cursor mode, use the Exit Cursor Addressing Mode routine.

You must be in the graphics mode to:

- Use general graphics input/output routines
- Set graphics attributes
- Draw generalized drawing primitives
- Format alpha text
- Control graphics text.

You cannot use any of the cursor control routines while in the graphics mode.

Cursor Mode

Select the cursor mode by using the Enter Cursor Addressing Mode routine. To exit the cursor mode and enter the graphics mode, use the Exit Cursor Addressing Mode routine. The cursor mode is only applicable to Cathode Ray Tube (CRT) devices.

You must be in the cursor mode to:

- Erase a full page, line, or part of a line

- Position output in any character cell

- Control video attributes such as blinking, boldface, underline, and reverse video.

You cannot use the graphics routines while in the cursor mode.

Toolkit Routines

The Toolkit routines are divided into the following functional groups:

- Workstation Control Routines
- Paging Routines
- Pel Routines
- Cursor Control Routines
- General Graphics Routines
- Graphics Primitives
- Graphics Text Routines
- Alpha Text Routines
- Input Routines.
- Error Handling.

Workstation Control Routines

The Workstation control routines control the flow of information to and from workstations. A workstation is an input and/or output graphics device. You need to open a workstation at the beginning and close a workstation at the end of a program that operates a workstation. The Control routines are:

- **Application Data.** Allows the Metafile Device Driver to place application specific data into the metafile (disk file containing graphics).

- **Clear Workstation.** Removes or erases information from a display, prompts for new paper on a plotter, and prints all pending graphics on a printer.

- **Close Workstation.** Prints all pending graphics or text, and then halts the flow of workstation information.

- **Hardcopy.** Generates a hardcopy (printout), if supported.

- **Message.** Places a text string in the metafile to be displayed by the Graphical File System as an operator message.

- **Open Workstation.** Prepares a workstation to receive or output information.

- **Set Pen Speed.** Sets the speed of the plotter pen.
- **Update Workstation.** Displays all pending text or graphics on the workstation.

Paging Routines

The Paging Routines govern the operation of pages. They are:

- **Copy Page.** Allows the application program to copy the contents of one page (active or nonactive) into another page.
- **Inquire Page.** Allows the application program to determine which page routine attributes are set.
- **Set Page.** Allows the application program to write to the specified page.

Pel Routines

The Pel routines move one or more pels (pixels) from one area on a display screen to another area or to memory. These routines work only in graphics mode. The Pel routines are:

- **Copy Pels.** Copies one or more pels from one position to another on the same display screen.
- **Get Pels.** Moves the pels into a storage array.
- **Put Pels.** Sends stored pels to the currently selected page.

Cursor Control Routines

The cursor control routines position the standard alphanumeric cursor, as well as place cursor text on the display screen. You can position the cursor on a display screen cell grid of rows and columns. A typical display screen measures 24 rows by 80 columns. The cursor control routines affect only CRT devices.

The cursor control routines are:

- **Cursor Down.** Moves the cursor down one row on the cursor cell grid.
- **Cursor Home.** Moves the cursor to the upper left corner (row 1, column 1).
- **Cursor Left.** Moves the cursor one column to the left.
- **Cursor Right.** Moves the cursor right one column.
- **Cursor Up.** Moves the cursor up one row on the cursor cell grid.
- **Direct Cursor Address.** Moves the cursor to a specified position on the display screen.
- **Enter Cursor Addressing Mode.** Prepares the display screen for cursor routines and places the cursor at home position.
- **Erase to End of Line.** Erases all text to the end of the line (row).
- **Erase to End of Screen.** Erases all text from the cursor position to the end of the display screen.
- **Exit Cursor Addressing Mode.** Exits the cursor-addressing mode if not in graphics mode.
- **Reverse Video Off.** Puts foreground color into the foreground and background color into the background.
- **Reverse Video On.** Puts foreground color into the background and background color into the foreground.
- **Set Cursor Text Attributes.** Sets the attributes of blink, bold, reverse video, and underline for subsequent cursor-addressable text.
- **Set Cursor Text Color Index.** Sets the foreground and background colors for the cursor-addressable text.
- **Set Cursor Text Mode.** Sets the mode to be used when the next Enter Cursor Addressing Mode routine call is received.

The following routines can be used to inquire about cursor control attributes or to write cursor text:

- **Inquire Addressable Character Cells.** Returns the particular format of a display screen to your program.
- **Inquire Current Cursor Text Address.** Returns the current cursor position to your program.
- **Inquire Cursor Text Mode.** Returns the mode to be used when the next Enter Cursor Addressing Mode routine call is received.
- **Output Cursor Addressable Text.** Writes text at the current cursor position and moves the cursor one space to the right.

General Graphics Routines

The general graphics routines are used to set color and writing modes, and to display or remove the graphics cursor, which is usually a crosshair. The general graphics routines are:

- **Display Graphic Input Cursor.** Displays the graphics cursor at a specified location on the workstation.
- **Remove Graphic Input Cursor.** Removes the graphics input cursor from the workstation.
- **Set Background Color Index.** Sets the background color on a workstation.
- **Set Color Representation.** Changes the color representation of a color index.
- **Set Graphic Color Burst Mode.** Sets the mode that the workstation uses when the next Exit Cursor Addressing Mode routine is received.
- **Set Writing Mode.** Controls how output writes over pending graphics in a print buffer or on the display screen.

The following routines can be used to inquire about the attributes set by general graphics routines:

- **Inquire Color Representation.** Returns information about the color table.
- **Inquire Graphic Color Burst Mode.** Returns the mode the workstation will use when the next Exit Cursor Addressing Mode routine is used.

Graphics Primitives

The VDI graphics routines cause graphics images, called primitives, to be displayed on workstation surfaces. These primitives are modified by attributes. Attributes are associated with:

- **Workstations.** Workstation attributes are specific to a workstation. They can be changed at any time and affect all displayed primitives.
- **Primitives.** Each type of primitive has its own set of attributes. For example, some attributes, such as line style, have meaning for the primitive polylines but not for filled areas. Other general attributes, such as color, can be set for all primitives.

Primitive attributes can be changed under program control.

Polyline Primitive and Attributes

A polyline primitive is a series of connected line segments. It is specified by giving the point coordinates of each of its vertexes.

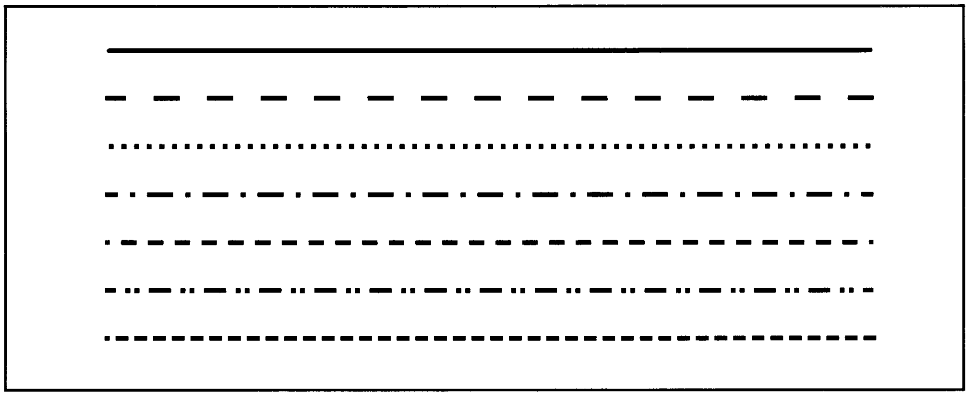


Figure 1-2. Polylines

Polyline attributes are:

- **Set Polyline Color Index.** Selects polyline color from current indexes.
- **Set Polyline Line Type.** Selects a line type from among those shown in Figure 1-2 or from a larger set, if additional types are supported by the selected device.
- **Set Polyline Line Width.** Controls the width of polylines.

The following routines can be used to inquire about the attributes of the polyline or to output the currently selected polyline:

- **Inquire Current Polyline Attributes.** Returns all current polyline features to your application program.
- **Output Polyline.** Draws a line (one or more line segments) on a workstation.

Polymarker Primitive and Attributes

A polymarker is a series of marker symbols drawn at specified points.

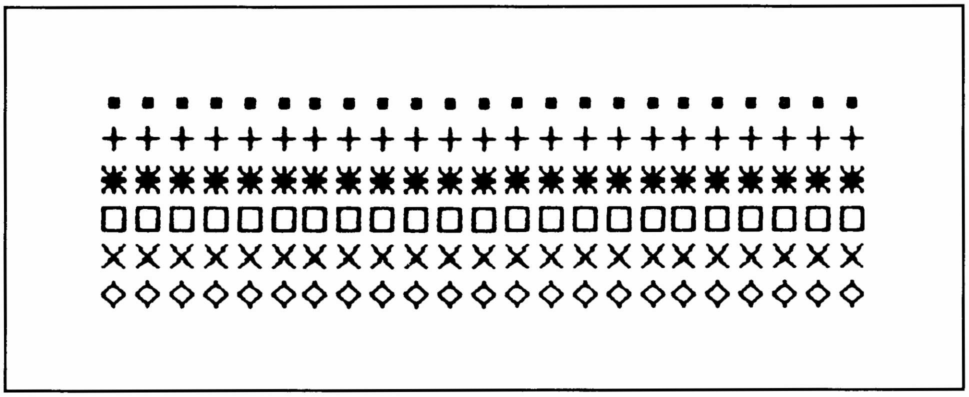


Figure 1-3. Polymarkers

Polymarker attributes are:

- **Set Polymarker Color Index.** Selects polymarker color from current indexes.
- **Set Polymarker Height.** Controls the height of polymarkers.
- **Set Polymarker Type.** Selects a polymarker type from among those shown in Figure 1-3 or from a larger set, if additional types are supported by the selected device.

Use these routines to inquire about polymarker attributes or to draw polymarkers:

- **Inquire Current Polymarker Attributes.** Returns the current polymarker features to your application program.
- **Output Polymarker.** Displays markers on a workstation.

Fill Area Primitive and Attributes

A fill area primitive is the interior of an arbitrary polygon, circle, pie slice, or bar that is painted with a color or pattern. It is specified by the vertexes of the shape that encloses it.

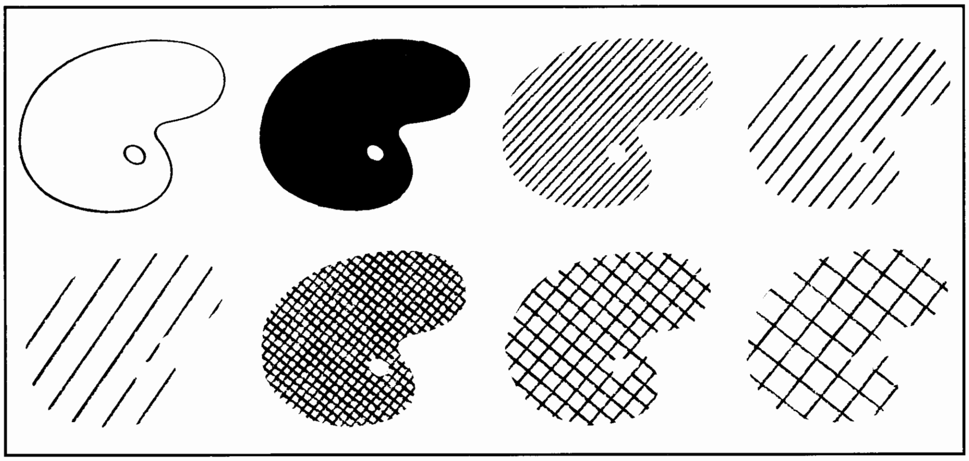


Figure 1-4. Fill Areas

Fill Area attributes are:

- **Set Fill Color Index.** Specifies the fill color by selecting a color index.
- **Set Fill Interior Style.** Determines the fill style (hollow, solid, pattern, or hatch).
- **Set Fill Style Index.** Selects one of the available patterns or hatches (ignored when a hollow or solid interior style has been specified).

Use the following routines to inquire about fill area attributes or output a fill area.

- **Inquire Current Fill Area Attributes.** Returns the current fill area attributes.
- **Output Filled Area.** Produces a filled area on the workstation.

Generalized Drawing Primitives

The VDI provides drawing capabilities that produce arcs, circles, bars, and other graphics objects. Use the following routines to inquire about Generalized Drawing Primitives or produce primitives:

- **Inquire Cell Array.** Returns the color indexes used to produce a specific cell array.
- **Output Arc.** Draws an arc on a workstation.
- **Output Bar.** Produces a rectangular area on a workstation using fill area attributes.
- **Output Cell Array.** Produces a rectangular area divided into one or more cells of equal size and shape. Each cell can have a unique color.
- **Output Circle.** Draws a circle on a workstation using fill area attributes.
- **Output Pie Slice.** Draws a pie slice on a workstation using fill area attributes.

Text Routines

The VDI routines enable you to output three kinds of text, as follows:

- **Alpha text.** High quality text suitable for letters and documents (typically used on high quality printers). This text is available only in graphics mode.
- **Graphics text.** Text usually used to label charts and diagrams or whenever large size text is needed. This text is available only in graphics mode.
- **Cursor text.** This text is available only in cursor mode. Refer to "Cursor Control Routines" in this chapter for a list of cursor text routines.

The following table shows the features available with the three types of text supported by the VDI:

Text Capabilities			
Text Attributes	Alpha Text Routines	Graphics Text Routines	Cursor Text Routines
Addressability	On NDC units	On NDC units	On characters, rows, and columns
Superscripting	Yes	No	No
Subscripting	Yes	No	No
Scaling	No	Yes	No
Rotation	No	Yes	No
Color	Yes	Yes	Yes
Reverse Video	No	No	Yes
Underlining	Yes	No	
Blinking	No	No	Yes
Bold Font	Yes	No	Yes
Variable Line No Spacing	Yes	Yes	
Overstriking	Yes	No	No
Pass-through No Mode	Yes	No	
Selectable No Quality	Yes	No	
Variable Text Height	Yes	Yes	No
Text Alignment	No	Yes	No

Graphics Text Routines

The graphics text routines set the height, alignment, and other text conditions used in your application program. The graphics text routines are:

- **Set Character Height.** Sets the height of graphics text characters.
- **Set Graphic Text Alignment.** Sets graphics text horizontal and vertical alignment.
- **Set Graphic Text Color Index.** Sets the graphics text color index.
- **Set Graphic Text Font.** Selects the text font for graphics text.
- **Set Graphic Text String Baseline Rotation.** Sets the baseline rotation of a string of graphics text characters.

Use the following routines to inquire about current graphics text attributes and to output graphics text.

- **Inquire Current Graphic Text Attributes.** Returns all current attributes that affect graphics text to your application program.
- **Output Graphic Text.** Writes graphics text on a workstation.

Alpha Text Routines

The Alpha text routines write document-quality text on a workstation, such as a printer. Alpha text is displayed according to the resolution and accuracy of the device. Alpha Text routines are:

- **Set Alpha Text Color Index.** Selects the color of subsequent alpha text.
- **Set Alpha Text Font and Size.** Sets the hardware font and size for subsequent alpha text output.
- **Set Alpha Text Line Spacing.** Sets the vertical spacing between lines of alpha text.
- **Set Alpha Text Overstrike Mode.** Turns overstriking on or off.
- **Set Alpha Text Pass Through Mode.** Turns pass-through mode on or off.

- **Set Alpha Text Position.** Sets the position of a string of alpha text characters on the workstation.
- **Set Alpha Text Quality.** Sets the alpha text quality level.
- **Set Alpha Text Subscript/Superscript Mode.** Causes alpha text to be offset below or above the text line.
- **Set Alpha Text Underline Mode.** Turns alpha text underlining on or off.

Use the following routines to inquire about current alpha text attributes and to output alpha text.

- **Inquire Alpha Text Capabilities.** Returns information regarding the alpha text features of the workstation used.
- **Inquire Alpha Text Cell Location.** Returns the location of the requested alpha text character cell.
- **Inquire Alpha Text Font Capability.** Returns features of a particular alpha text font and size.
- **Inquire Alpha Text Position.** Returns the current alpha text position to your program.
- **Inquire Alpha Text String Length.** Returns the length of the alpha text string, based on the current font in use.
- **Output Alpha Text.** Writes the alpha text string at the current alpha text position.

Input Routines

The input routines supply graphics information to your application program. Input routines operate in the following modes:

- **Request mode.** The application program user must signal when the data is ready to be entered by pressing a key or button.
- **Sample mode.** Pending input is returned to the application program immediately.

The Input routines are:

- **Input Choice (request mode).** Activates the choice device (such as function keys) and waits for a selection before returning to the program.
- **Input Choice (sample mode).** Polls the choice device. If a choice is pending, it is returned.
- **Input Locator (request mode).** Causes the graphics cursor to be displayed on a workstation until some operator interaction has taken place.
- **Input Locator (sample mode).** Returns the current position of the graphics input cursor without waiting for operator interaction.
- **Input String (request mode).** Accepts character input from the keyboard and waits for the input before proceeding with the application.
- **Input String (sample mode).** Polls the keyboard of the system.
- **Input Valuator (request mode).** Activates the valuator (potentiometer) device; the user sets it to the desired value.
- **Input Valuator (sample mode).** Returns the current value of the valuator device without waiting for operator interaction.
- **Read Cursor Movement Keys.** Determines if a cursor movement key was struck.
- **Set Line Edit Characters.** Sets the current line editing characters.

Error Handling

There is only one routine associated with error handling. This routine, Inquire Error, must be used to acquire the actual error code, after some other routine has returned a status of -1, indicating that an error has occurred. Refer to Appendix D, “Error Codes”, for more information about error codes.

Workstations

The following text contains a brief description of a workstation and lists the Toolkit routines that are associated with workstation control.

What are Workstations

A workstation is a computer peripheral that is connected to a host system. When the host "opens" a workstation, that workstation is assigned a unique ID number and a type identifier. After this point, the VDI software running on the host accesses a workstation via this unique ID number. Note that more than one workstation can be open simultaneously.

The VDI supports the following four types of workstations:

- Input
- Output
- Input and Output
- Metafile output.

Workstation Control Routines

The Workstation control routines are a set of routines that perform several important system and workstation functions, including the following:

- Open Workstation
- Update Workstation
- Clear Workstation
- Close Workstation
- Escape Routines.

Open Workstation

The Open Workstation routine must be the first Toolkit routine called by an application program. This routine establishes several essential parameters for the workstation and loads the required device driver into memory.

Update Workstation

A workstation can be updated with the Update Workstation routine. This causes all buffered output to be sent to the workstation's display area.

Clear Workstation

A workstation is cleared with the Clear Workstation routine. This clears or erases the display surface, if not already empty. It also prints pending graphics on a printer, clears the printer buffer, and advances the paper. On a plotter, all pending graphics are printed and the user is prompted for a paper change.

Close Workstation

A workstation is closed with the Close Workstation routine. This prints all pending graphics or text, and then halts the flow of workstation information.

The Close Workstation routine must be called prior to program termination to avoid unpredictable results.

Escape Routines

Escape routines are VDI routines that allow the programmer to perform non-standard graphics routines with the workstation. Use these routines to take advantage of special capabilities of workstations.

Escape routines are device-specific routines that do not work on all workstations. For example, the Set Pen Speed routine only affects a plotter, and the Cursor Right routine only affects a CRT device.

Chapter 2. Programming Considerations

CONTENTS

About this Chapter

This chapter describes some programming considerations for the Graphics Development Toolkit. The chapter includes a discussion of the following topics:

- Programming with the Toolkit
- Helpful Hints.

For information on programming considerations that involve specific languages, refer to the language reference booklets that accompany the Toolkit.

Programming with the Toolkit

The Graphics Development Toolkit is used to write graphics programs. You need to understand certain graphics programming concepts before you begin.

Before Writing Your Program

Before you begin writing your program, you need to understand the concepts of Normalized Device Coordinates (NDC) and the Aspect Ratio of coordinates.

Normalized Device Coordinates (NDC)

The following contains both a description of NDC Coordinates and a brief explanation of how to use them.

What are NDC Coordinates

Each point (NDC coordinate) on a workstation is individually addressable in your application program. When your program runs, VDI maps the Normalized Device Coordinates (NDC) to pels (pixels) or device units of the workstation, rounding off where needed.

Using NDC Units

When you write your program, think of NDC units as percentages of the actual device coordinates for each workstation. Ten percent of the actual device coordinates represent 10% of the NDC coordinates used by the application. For example, in a 32K X 32K NDC space, an NDC coordinate of x=16348 and y=0 is located on the workstation in the middle (the x axis) at the bottom (the y axis).

Also, a horizontal line 3276 NDC units long (used by the application) is represented by a line that is 10% of the length of the x axis on the workstation. If the workstation has 100 pels on its x axis, the line will be 10 pels long. If the x axis has 250 pels, the line will be 25 pels long.

Aspect Ratio

The following contains both a description of Aspect Ratio and a brief explanation of how to control it.

What is Aspect Ratio

The aspect ratio is the ratio of the horizontal to vertical dimensions of an image. In the VDI you can preserve that ratio on all surfaces.

Controlling Aspect Ratio

There are two user-selectable modes in which NDC units to device coordinate transformation can take place.

In the first mode, (aspect ratio not preserved) the NDC range of 0-32767 is mapped to the full extent of the physical workstation surface on each axis. Using this mode insures that all the graphics information will appear on the workstation surface since all NDC points are displayable.

The following example shows an arrow with a height of 8192 NDC units, and aspect ratio not preserved:

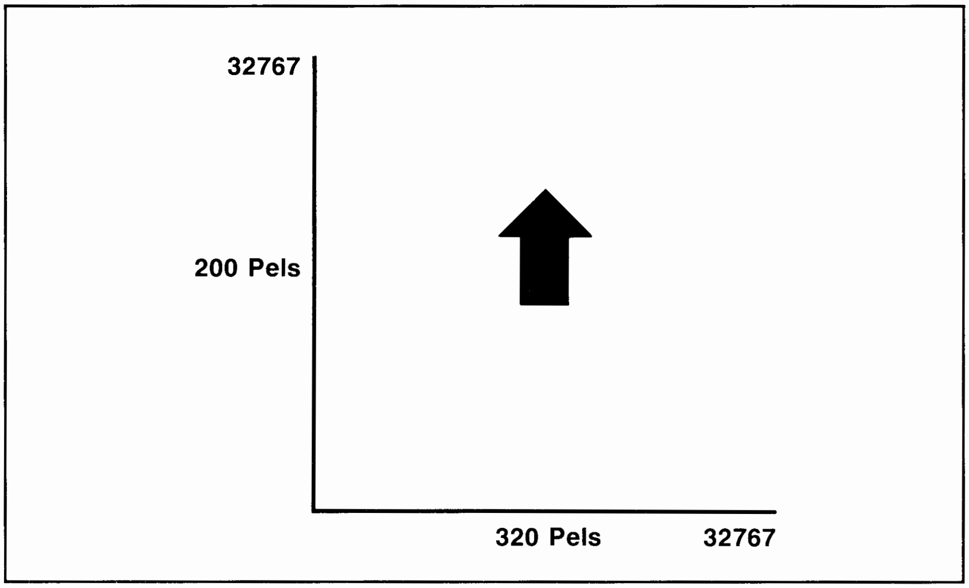


Figure 2-1. Aspect Ratio, Not Preserved

Distortion can occur if the workstation does not map NDC units to equal physical distances in both directions (this happens on workstations with non-unity device units). The result is that squares turn into rectangles.

To avoid this situation, use the second mode (aspect ratio preserved). This mode preserves the aspect ratio of the image by mapping NDC units to equal physical distances on both axes. To do this, the full NDC space is mapped into the longest axis of the workstation. The other axis displays as much of the NDC space as possible, but some information at one edge is lost.

The following example shows an arrow with the same NDC coordinates as the first arrow, but with aspect ratio preserved (the arrow is now elongated):

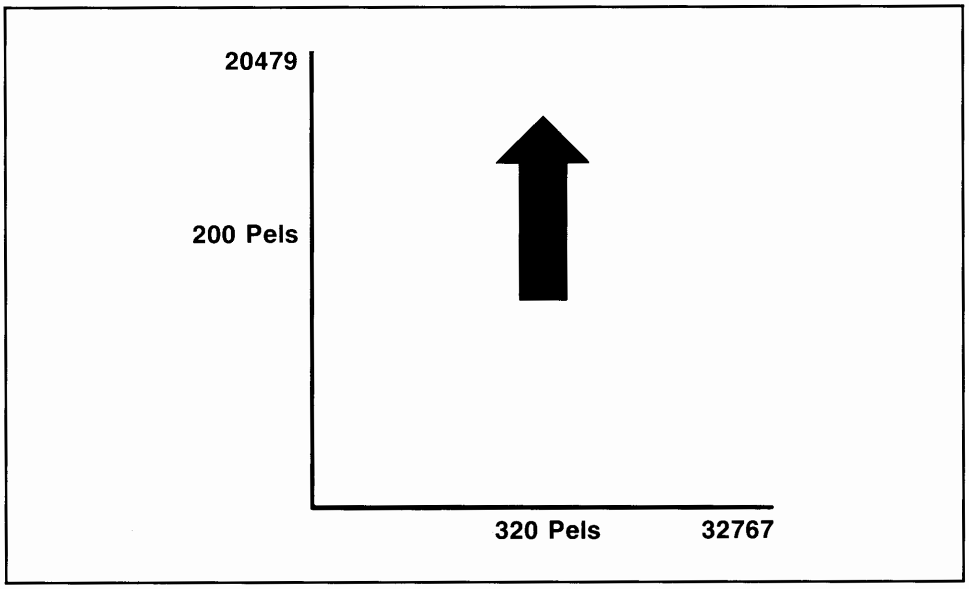


Figure 2-2. Aspect Ratio, Preserved

Compensation is provided in VDI for workstations with non-square pels so that circles appear as circles and squares look like squares. The application program is responsible in this case for sending only displayable NDC units to the system. However, device drivers automatically take workstation dependencies into account when using the bar, pie slice, arc, and circle primitives.

The non-preserved mode unburdens the application from doing a specific workstation-dependent transform. The advantage of using the preserved mode is that pictures can be easily transported between workstations with the assumption that unity (square) aspect ratio is used. VDI will make the adjustment for the actual aspect ratio of the workstation.

The aspect ratio is controlled by the value you choose for the first element of the Open Workstation *workin* array (see the "Open Workstation" routine in Chapter 3 of this manual).

Writing Applications

You can install VDI on your system before or after you write your application, but you must install it before your application program is compiled. Refer to Appendix A, "Installing the Graphics Development Toolkit," for information about installing the Toolkit.

Write your application program to include calls to the VDI routines. Refer to the language reference booklets for the routine's syntax and parameter sequences of your particular programming language.

An application program usually consists of the following steps:

Step 1. Open Workstation

The Open Workstation routine is the first VDI routine invoked in an application program. It defines the type of workstation used, loads the device driver into memory, and returns information regarding the capabilities of the workstation requested. This information includes:

- Device type
- Number of colors available
- Number of text sizes available
- Number of line styles available.

Step 2. Set Graphics Primitive Attributes

The appearance of the graphics primitives can be modified by setting their attributes. The attributes that can be set include:

- Polyline and polymarker types
- Polyline width
- Polymarker and text height
- Fill area styles
- Color
- Text font and alignment.

For additional information on setting attributes, refer to "Attribute Setting Hints" in this chapter.

Step 3. Output Graphics Primitives

The output graphics primitives can be displayed on workstation surfaces. Output graphics primitive options include:

- Polyline and polymarker types
- Fill area styles
- Graphics text
- Generalized drawing primitives.

Step 4. Input Graphics Primitives

A user can input information to an application program through the following logical devices:

- **Locator.** Selects a position on the display surface by moving a graphics input cursor or crosshairs to the desired position.

- **Valuator.** Returns a logical value that is a number corresponding to the condition of the physical input device, such as, the position of a dial.

- **Choice.** Provides values that are integers between 0 and a workstation-dependent maximum. This typically happens when the operator presses a button or function key.

- **String.** Returns a character string, typically from a keyboard.

Step 5. Inquiry Routines

The VDI includes inquiry routines that can inform your application program about the current state of the system, including:

- Current attribute settings
- Device capabilities
- Workstations.

Step 6. Error Handling

VDI provides an Inquire Error routine to use after other Toolkit routines return errors. Device drivers also return errors when the workstation is not capable of performing a routine.

Step 7. Close Workstation

Prints all pending graphics or text, and then halts the flow of workstation information.

Step 8. Set the Environment

Refer to Appendix A, "Installing the Graphics Development Toolkit", for information on setting the environment for the device that you are going to use for your application. Refer to Appendix C, "Graphics Drivers", for information about device options that must be set via the environment.

Step 9. Compile and Run Your Program

After you have incorporated all necessary VDI routines, compile and link your application program. For specific compiling and linking information, refer to the language reference booklets.

Your graphics application program is now ready to run.

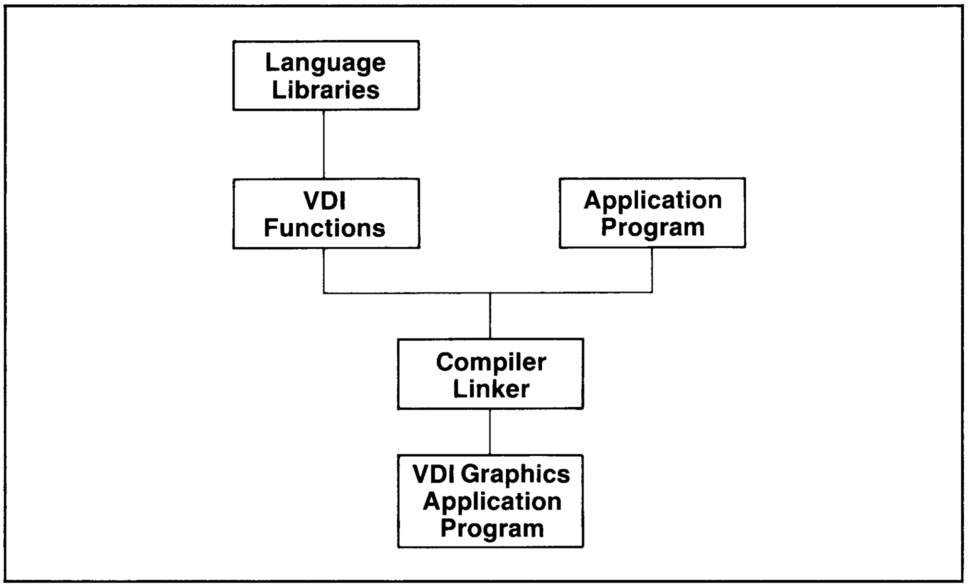


Figure 2-3. Software Relationships

Helpful Hints

The helpful programming hints are divided into groups. Each group contains hints related to that group. These hints should help you to learn and understand this product.

Control Hints

- Certain routines, if used in an application, make that application device-dependent. These routines are marked with a double asterisk (**) within Chapter 3, 'Toolkit Routines.' Avoid using these routines, if your application program must remain device-independent.
- When displaying graphics to a plotter, the application should set the prompt flag when opening the workstation so that a "change paper" prompt is presented to the user.
- The background color of a CRT is not changed after a Set Background Color Index routine until a Clear Workstation routine is executed.
- Background color cannot be redefined on printers and plotters.
- An application can divide the actual device coordinates up into its own units by using the two NDC values returned by the Open Workstation routine. The two values are the 52nd and 53rd elements of the Open Workstation *workout* array (see "Open Workstation" in Chapter 3).

The following example uses *workout(51)* and *workout(52)*, corresponding to the 52nd and 53rd elements of the *workout* array, to divide a display surface into percentages. This C language program contains two code segments. The first code segment defines two routines that convert percentages to NDC units for both the x and y-axis. The second code segment uses the routines defined in the first code segment.

```
/*

define functions that convert from percentages
to NDC units

*/


#define UNIT 100
```

```
short fnxper(a)
short a;
/*
this will convert the X coordinates from percentages
to NDC units
*/
{
   extern short workout[];
   return((a / UNIT) * workout[51]);
}


short fnyper(a)
short a;
/*
this will convert the Y coordinates from percentages
to NDC units
*/
{
   extern short workout[];
   return((a / UNIT) * workout[52]);
}
          .
          .
          .
          .

 temp1 = fnxper(10);
 temp2 = fnyper(80);
 /*

 output graphics text at location (temp1, temp2)

 */
 v_gtext(device_handle,
         temp1,
         temp2,
         "Graphic text string");
```

```
/*

The preceding call to 'Output Graphic Text'
will start displaying the text string
with an indentation of 10% from the left
margin and 80% from the bottom of the device.

*/
```

Note: Application programmers must use only those units that are applicable to their product.

One advantage of the previous technique is that the aspect ratio can be changed without affecting an application, if the routines are used to convert from application coordinates to NDC units. The reason for this is that the array *(workout)* returned when a workstation is opened has its value calculated with the aspect ratio.

This technique could have the disadvantage of slowing the application execution speed, because coordinate conversion must be performed each time NDC units are needed.

- Cursor addressing has no effect on printers. It does not return errors but rather the routines are ignored. The Inquire Cursor Addressable Text routine indicates that the cursor text is not available.

- To find out what mode (graphics or cursor) an application last set, call an Inquire Current Cursor Text Address routine. If a -3095 error is returned, the workstation is in graphics mode.

- Cursor-addressable text is mutually exclusive with graphics text and alpha text. Graphics text and alpha text are compatible and can be displayed simultaneously on the same workstation.

- Graphics text pages and cursor text pages are not always in separate buffers. With the IBM Enhanced Graphics Adapter, the cursor text pages and the graphics pages share the same buffer area. When entering cursor mode, the active cursor text page is cleared to blank spaces. When entering graphics mode, the active graphics page is cleared to the workstation's background color.

When cursor mode is entered, either the active cursor page or all cursor pages will be cleared, depending upon the features of a specific workstation. When graphics mode is entered, either the active graphics page or all graphics pages will be cleared, depending upon the features of a specific workstation. You should use the Clear Workstation routine, when changing the active page, to ensure that a page is cleared properly.

- If graphics text, graphics primitives, or alpha text is sent to a printer, the Update Workstation routine must be executed, before anything is actually printed (see the "Update Workstation" routine in Chapter 3). In addition, the Clear Workstation routine presents any pending graphics, to a printer, before clearing the workstation (see "Clear Workstation" in Chapter 3). The Close Workstation routine displays all pending graphics, before closing a printer workstation (see "Close Workstation" in Chapter 3).

- The Clear Workstation routine removes all pending graphics from a printer buffer after printing the graphics. The Update Workstation routine prints pending graphics, but does not remove the pending graphics from the printer buffer.

Note: Pending graphics are graphics written to printer devices that have not been Updated, Cleared, or Closed.

Attribute Setting Hints

- Attribute setting routines always return the attribute value selected. This is either the closest value to the one requested, or the specified default in cases where the requested value is out of range. For example, default line style is index 1 (solid)—out of range line types are mapped to 1.

- Color index attribute setting routines select the closest index to the one requested. This can have interesting side effects. If a negative color index is requested, the closest index selected is zero (0). However, the default value for color index 0 is black. This causes all subsequent primitives that are drawn using that color index to be invisible on the display screen, if the display background color is set to black.

Graphics Primitive Output Hints

- When using the aspect ratio preservation mode, scaling primitives may be partially or totally clipped, and therefore, not visible.
- Due to the mode of specification for circles and arcs (center point/radius), it is not possible to display a circle or arc whose center point is not on the workstation surface.
- Graphics text characters are aligned on the left top corner of the character cell grid, as shown in Figure 2-4.

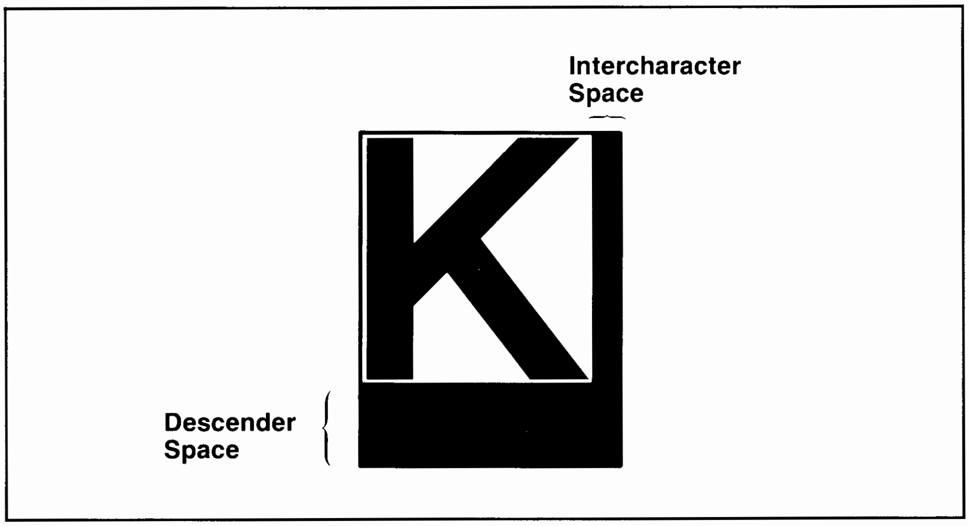


Figure 2-4. Character Grid Cell

- Filled areas are not outlined unless the fill interior style is hollow. To display a filled area (polygon, bar, circle, or pie slice) with an outline, first fill the area with the fill style of solid, pattern or hatch. Then fill the same area with fill interior style of hollow.
- The Set Graphic Text Alignment routine aligns the text on the character, not on the cell grid, as shown in Figure 2-5.

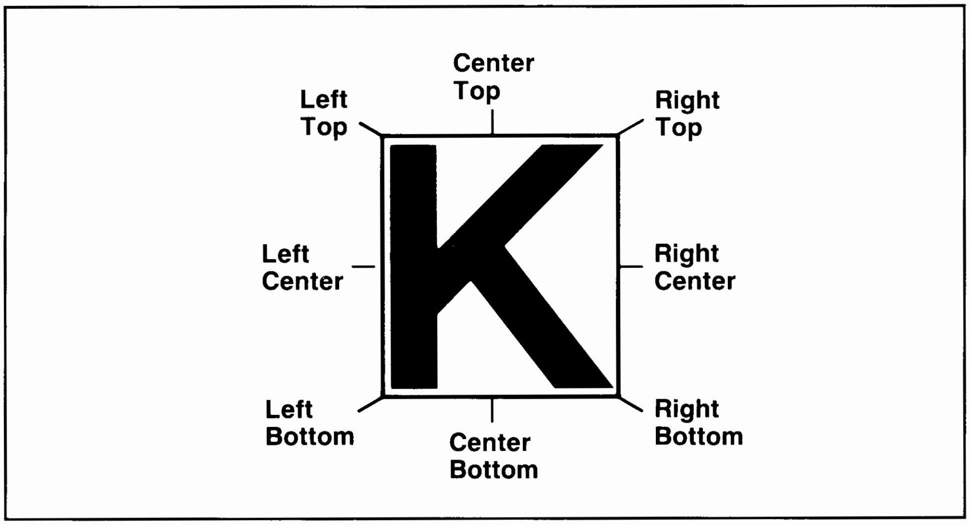


Figure 2-5. Character Alignment Points

- Graphics and alpha text characters are entirely clipped if any part of the character cell is not in the valid NDC range, as shown in Figure 2-6.

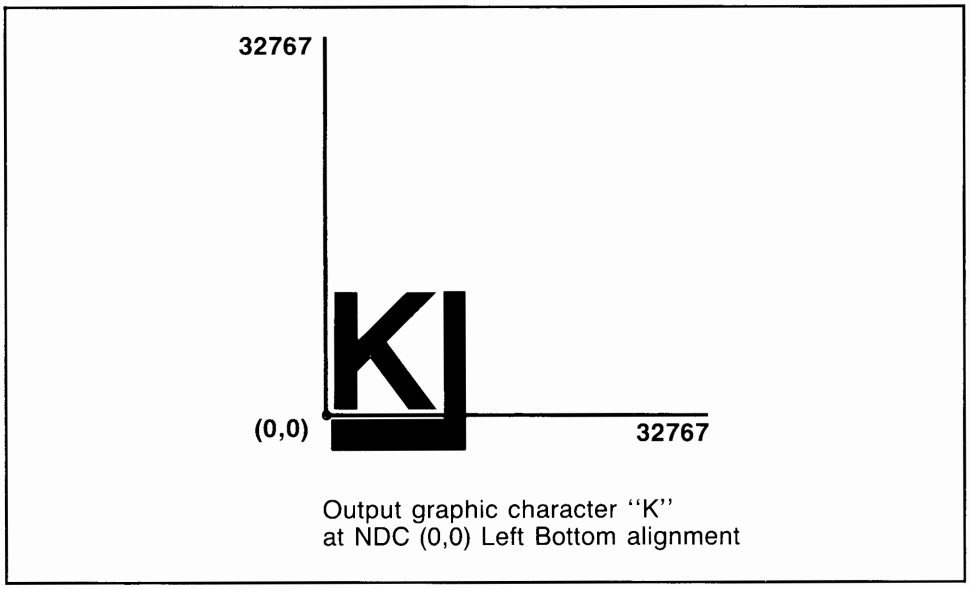

32767
(0,0)
32767
Output graphic character "K"
at NDC (0,0) Left Bottom alignment

Figure 2-6. Clipping

Notice in the above example that part of the cell is below the alignment point. For example, if this character is output at the NDC location (0,0) with left bottom alignment, the text is not displayed because part of the cell grid is out of the display region.

- The radius of a circle, arc, and pie slice is computed along the x-axis. All circles, arcs, and pie slices drawn look round on the workstation surface if aspect ratio is on or off. When the aspect ratio is not preserved, the radius does not stay the same NDC units, when drawing from the start angle to the end angle.

The following example shows a circle drawn at the coordinates (16384, 16384) with aspect ratio not preserved:

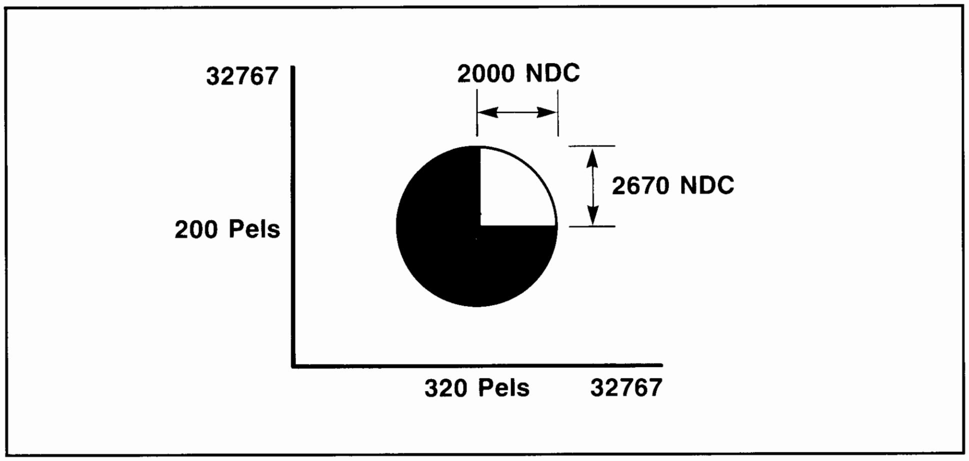


Figure 2-7. Circle, Non-Preserved Ratio

For a workstation with the maximum x value of 32767 and y value of 20479, the radius in NDC units stays the same when a circle is drawn with aspect ratio preserved. Figure 2-8 shows a circle drawn with aspect ratio preserved.

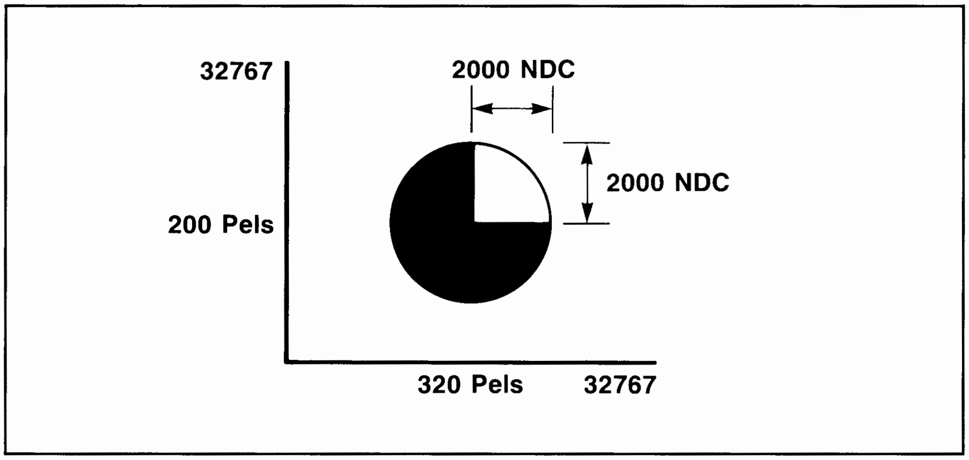


Figure 2-8. Circle, Preserved Ratio

Input and Inquiry Hints

- Routines that return arrays of data have the potential to write over application data if the amount of space allocated is smaller than the size given to the VDI. Make the input string as large as possible to avoid writing over your data area.
- When working with alpha text that will be displayed to a printer, an application developer should be aware that the following modes have no inquire routines:
  - Overstrike Mode ON/OFF
  - Pass-Through Mode ON/OFF
  - Quality Mode ON/OFF.

This should be a concern when writing alpha text to a printer; an application could run slower if it is required to set attributes of a printer many times. If an application needs to inquire about the above modes, it should keep its own inquire table for them.

- You can set the pen speed for a plotter, but you cannot inquire about it. The Set Pen Speed routine returns the actual pen speed set on the plotter. The Set Pen Speed routine affects a pen's speed only when your output is text or polylines. Fill areas are drawn at a constant speed set by VDI.
- The current writing mode can be found by using: the Inquire Current Fill Area Attributes routine, the Inquire Current Polyline Attributes routine, the Inquire Current Polymarker Attributes routine, or the Set Writing Mode routine.

Invalid Value Hints

- Values passed to VDI routines should be range checked (ensuring the value is in the valid range) in some cases to avoid unpredictable application program failures. If an x/y-coordinate is specified out of the valid NDC range, unpredictable program results occur. The NDC range changes depending on whether aspect ratio is preserved or not.

VDI does not provide range checking for the Open Workstation routine. If an invalid value is input into the *workin* array, the workstation might not open.

Error Handling Hints

- An error routine should be called after each VDI call. An example of an error would be "Illegal device handle" which would be returned if the workstation is already open. A C language example that calls "Output Graphic Text" follows:

```
/* percentage conversion functions */
extern short fnyper(), fnxper();
static char out_string[] = "Graphic text string";
short temp1, temp2;

temp1 = fnxper(15);
temp2 = fnyper(80);
```

```
/*

output graphics text at location (temp1, temp2)

*/

if( v_gtext(device_handle,
            temp1,
            temp2,
            out_string ) < 0 )
   vqerror();
   .
   .
   .


/*

this function will display an error

*/
vqerror()
{
   extern short vq_error();

   printf("Error number %d\n\r", vq_error());
}
```

Chapter 3. Toolkit Routines

CONTENTS

About this Chapter

This chapter contains a description of each routine that is available with the IBM RT PC Graphics Development Toolkit. The descriptions are intended to provide you with information about the purpose and general operation of a routine; they do not contain information about specific language syntax. You must refer to the language reference booklets, that are a part of the Toolkit, for information about the actual syntax of a routine.

The chapter is divided into several sections. Each section contains a group of routines that are associated with some general purpose or area of functionality. Within a section, the routines appear in alphabetical order. The sections are:

- Workstation Control Routines
- Paging Routines
- Pel Routines
- Cursor Control Routines
- General Graphics Routines
- Graphics Primitives
- Graphics Text Routines
- Alpha Text Routines
- Input Routines
- Error Handling.

Note: The routines shown in the language reference booklets are presented in alphabetical order; not divided into functional groups as they are in this chapter.

How Routines are Described

For each routine, information has been organized into six topics, as follows:

Purpose: The "Purpose:" entry for a routine describes the function or purpose of that routine.

Format: The "Format:" entry for a routine shows the name of that routine followed by both its input and output parameters, in parentheses.

> Inquire Current Cursor Text Address**
> (handle, row, column)

In the above example, "Inquire Current Cursor Text Address" is the name of the routine, *handle* is an input parameter, and both *row* and *column* are output parameters.

Note: If a double asterisk (**) follows the name of a routine, as in the above example, the operation of that routine is device-dependent. Using these routines in an application program makes that application device-dependent. Thus, you must use these routines only during program development, or in an application that does not require device-independence.

Input: The Input parameters are arrays or variables to which you must assign values, before calling the VDI routines. These parameters are shown first within the entry for the "Format:" of a routine.

Output: Output parameters are arrays or variables that receive values from the VDI routine that is called by your application program. These parameters are listed after the input parameters within the "Format:" entry for a routine.

Status: Each routine returns a status or error code, to the application program, when that routine is completed.

Remarks: The "Remarks:" entry for a routine contains information which is particularly important to understanding the operation of a routine. It includes qualifications and notes about the effects the routine being described has on other routines, the VDI, and the general operation of the Graphics Development Toolkit.

Note: Variables that are arrays are described in this book as having a given number of elements. In the language reference booklets, there is a definition of array data types for each array.

Workstation Control Routines

This section describes the device control routines of the VDI. These routines control the flow of information to and from a workstation (an input and/or output graphics device). You must include the Open Workstation routine at the beginning and the Close Workstation routine at the end of any application program that operates a workstation.

Routines included in this section are:

- Application Data
- Clear Workstation
- Close Workstation
- Hardcopy
- Message
- Open Workstation
- Set Pen Speed
- Update Workstation.

Application Data

Purpose: This routine allows the Virtual Device Metafile driver to place application-specific data in the metafile (disk file containing graphics).

Format: Application Data**
(handle, routine name, length, data)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

routine name
A text string.

length
Number of elements of application data.

data
An array containing application data.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine places application data in a metafile. The information is available to an application program via the Get Metafile Item routine in the IBM RT PC Graphical File System. The *routine name* is a user-defined title for whatever the application data element represents.

For information on the IBM RT PC Graphical File System or the Get Metafile Item routine, refer to the *IBM RT PC Graphical File System Programmer's/User's Guide*.

Clear Workstation

Purpose: Removes or erases graphics information from the workstation.

Format: Clear Workstation
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when one or more workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine has different effects on different types of devices:

Printers: All pending graphics are printed, paper is advanced to a new sheet, and the print buffer is cleared.

Plotters: All pending graphics are displayed and a prompt appears on the screen to change the paper. You can control the display of prompts in the 11th element of the Open Workstation *workin* array (see "Open Workstation").

CRT Screens: The screen is cleared (to the current background color). On color screens this routine must be called after the Set Background Color Index routine if the color is to change. The background color is not changed until a Clear Workstation routine is executed.

Close Workstation

Purpose: Prints all pending graphics or text, and then halts the flow of workstation information.

Format: Close Workstation
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The Close Workstation routine must be the last IBM RT PC Graphics Development Toolkit routine called by an application program.

Hardcopy

Purpose: Generates a hardcopy (printout).

Format: Hardcopy**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine is device-specific and can involve copying the display screen to a printer.

This routine is not supported by any of the device drivers in the IBM RT PC Graphics Development Toolkit.

Message

Purpose: Places a text string in the metafile to be displayed by the IBM RT PC Graphical File System as an operator message.

Format: Message**
(handle, message, pause indicator)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

message
A text string.

pause indicator
Determines whether the IBM RT PC Graphical File System will pause for a response from the user, when the Interpret Metafile Item routine is used.

0 = No response required

1 = Pause after issuing message and wait for a response

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine causes the "message" to appear on a workstation's display area in some device-dependent position when the metafile is interpreted. A specific display may be selected by the MESSAGEPORT environmental parameter. Refer to Appendix A, "Installing the Graphics Development Toolkit", for information about setting the *MESSAGEPORT* environmental parameter.

For information on the IBM RT PC Graphical File System, refer to the *IBM RT PC Graphical File System Programmer's/User's Guide*.

Open Workstation

Purpose: Prepares a workstation to receive information.

Format: Open Workstation
(workin, handle, workout)

Input: **workin**
A 19-element array containing the identification of the workstation, font styles, color indexes, and style indexes.

All nineteen elements must be assigned initial values before the Open Workstation routine can open a device successfully. Assign values to each element using the values found in Appendix C, "Graphics Drivers."

workin(1)
Coordinates transformation mode flag. Determines how to transform NDC units to device coordinates:

0 = Map NDC units to the full extent of each axis. This does not preserve the aspect ratio. Picture will fill entire display screen.

1 = Map NDC units to the full extent of the longest axis only; map subset of NDC to shorter axis. This mode preserves the unity aspect ratio. Using this technique and the proper scaling factor results in a picture with the same aspect ratio.

workin(2)
Polyline line type.

workin(3)
Polyline color index.

workin(4)
Polymarker type.

workin(5)
Polymarker color index.

workin(6)
Graphics text font.

workin(7)
Graphics text color index.

workin(8)
Fill interior style.

workin(9)
Fill style index.

workin(10)
Fill color index.

workin(11)
Prompting flag for controlling the screen prompts (for paper and pen changes on plotters):

0=Do not display device-dependent prompts to the logical message device.

1=Display device-dependent prompts to the logical message device.

workin(12-19)
Device driver logical name. This is an ADE (ASCII Decimal Equivalency) form that is used to determine which environmental parameter is used to locate the device driver. Programming examples included in the language reference booklets show how to code the logical name in ADE into the *workin* array. See "Setting Environmental Parameters" in Appendix A for a description of how to assign the logical device name to the actual device driver name.

Output: **handle**
The device ID associated with the workstation identifier *(workin(12)* through *workin(19))*. Give this variable a descriptive name for the device you are opening. Use the variable name to identify the device you want to be affected by a routine. Always choose a different variable name for each device you open.

workout
A 66-element array in which the Open Workstation routine returns device information.

workout(1)
Maximum addressable width of screen/plotter in rasters/steps assuming a 0 starting point (for example, a resolution of 640 implies an addressable area of 0-639, so *workout(1)* would be 639).

workout(2)
Maximum addressable height of screen/plotter in rasters/steps assuming a 0 start point (for example, a resolution of 480 implies an area of 0-479, so *workout(2)* would be 479).

workout(3)
Device coordinate units flag:

0=Device capable of producing a precisely scaled image (typically plotters and printers).

1=Device not capable of a precisely scaled image (CRTs).

workout(4)
Width of one pel (plotter step) in micrometers.

workout(5)
Height of one pel (plotter step) in micrometers.

workout(6)
Number of character heights (0=continuous scaling).

workout(7)
Number of line types (0=device is not capable of graphics).

workout(8)
Number of line widths.

workout(9)
Number of marker types.

workout(10)
Number of marker sizes (0=continuous scaling).

workout(11)
Number of graphics text fonts.

workout(12)
Number of patterns.

workout(13)
Number of hatch styles.

workout(14)
Number of predefined colors (at least 2 even for monochrome device). This is the number of colors that can be displayed on the device simultaneously.

workout(15)
Number of Generalized Drawing Primitives (GDP).

workout(16-25)
List of GDPs (up to 10 allowed):

-1 = No GDP
1 = bar
2 = arc
3 = pie slice
4 = circle

workout(26-35)
Attribute set associated with each GDP:

-1 = GDP does not exist
0 = Polyline
1 = Polymarker
2 = Text
3 = Fill area
4 = None
5 = Other

workout(36)
Color capability flag:

0 = No
1 = Yes

workout(37)
Text rotation capability flag:

0 = No
1 = Yes

workout(38)
Fill area capability flag:

0 = No
1 = Yes

workout(39)
Pel operation capability flag:

0 = No
1 = Yes

workout(40)
Total number of colors the workstation can display. This may be larger than the number of colors the device can display simultaneously:

0 = Continuous device
2 = Monochrome (black and white)
>2 = Number of colors available

workout(41)
Locator capability flag:

0 = No
1 = Yes

workout(42)
Valuator capability flag:

0 = No
1 = Yes

workout(43)
Number of choices available (1 to n).

workout(44)
String input capability flag:

0 = No
1 = Yes

workout(45)
Workstation type:

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

workout(46)
Device type:

0=CRT
1=Plotter
2=Printer
3=reserved
4=Metafile output
5=Other

workout(47)
Number of writing modes available.

workout(48)
Highest level of input mode available:

0=None
1=Request
2=Sample

workout(49)
Text alignment capability flag:

0=No
1=Yes

workout(50)
Inking capability flag as output echo device:

0=No
1=Yes

workout(51)
Rubberbanding capability flag as an output echo device:

0=No rubberband capability
1=Capable of rubberband lines
2=Capable of rubberband lines and rectangles

workout(52)
Maximum addressable NDC unit coordinate on x-axis. This value is filled in based on the coordinate transformation mode selected.

workout(53)
Maximum addressable NDC unit coordinate on y-axis. This value is filled in based on the coordinate transformation mode selected.

workout(54-58)
Version of the driver. This is an ADE character string that represents the version of the driver in the following form: vv.ll where vv is the actual version and ll is the level.

workout(59-60)
Reserved.

workout(61)
Minimum graphics character height in NDC units.

workout(62)
Maximum graphics character width in NDC units.

workout(63)
Minimum line width in NDC units.

workout(64)
Maximum line width in NDC units.

workout(65)
Minimum marker height in NDC units.

workout(66)
Maximum marker height in NDC units.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see “Inquire Error”).

Remarks: This routine initializes a workstation. It sets all defaults and returns device information. Both the alpha and graphics display surfaces are cleared by this routine. You can change default values set by using individual routines.

All 19 elements of the *workin* array must be assigned initial values before the Open Workstation routine can open a device successfully. Assign values to each element using the values found in Appendix C, "Graphics Drivers."

You can open multiple workstations at any time.

A device driver must be assigned to the logical name used when the open occurs. See "Setting Environmental Parameters" in Appendix A.

Open Workstation Default Values

The default values set by the Open Workstation routine are given in the following list:

- Graphics mode is on. In this mode, you can use any routine except cursor-addressing routines.
- Graphics character size is the largest that can fit on a cell grid of 24 by 80 characters.
- Character baseline rotation=0 degrees. To change this, see "Set Graphic Text String Baseline Rotation."
- Line width=1 device unit.
- Marker height=minimum device height.
- Writing mode=4 (replace) for all devices except plotters.
- Writing mode=8 (overstrike) for plotters only.
- Input mode=request for all inputs.
- Graphics text alignment=bottom for vertical text and left for horizontal text.
- Cursor-addressing mode is off.
- Alpha text position=upper left corner of a workstation.
- Alpha text line spacing=single.
- Alpha text font=standard font for a 24 by 80 character cell grid.

- Alpha text subscripting and superscripting=off.
- Alpha text underlining=off.
- Alpha text overstriking=off.
- Alpha text pass-through=off.
- Alpha text quality=highest (100%).
- Alpha text color=white for screens and black for printers.
- Line delete character is **Ctrl_U** or ASCII NAK (ASCII value 21).
- Character delete character is **Ctrl_H** or ASCII Backspace (ASCII value 8).
- **Color.** All colors are set by using the default color tables shown in Figure 3-1. Each color has an associated color index (number). The color associated with an index can be changed by using a Set Color Representation routine (see "Set Color Representation"). For device specific color information, see Appendix C, "Graphics Drivers."

Default Color Table	
Index	Color
0	Black for screens, white for printers and plotters
1	White for screens, black for printers and plotters
2	Red
3	Green
4	Blue
5	Yellow
6	Cyan (blue-green)
7	Magenta (blue-red)
〉7	White

Figure 3-1. Default Color Table

Set Pen Speed

Purpose: Sets the speed of the plotter pen.

Format: Set Pen Speed**
(handle, speed)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

speed
Pen speed as a percentage of maximum speed (1-100).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

)0=Realized pen speed.

0=No error.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see “Inquire Error”).

Remarks: This routine affects only plotter devices. The routine can be used to cause a plotter to print polylines or text more slowly, when you are using nonstandard inks or media. Fill areas are filled at a constant speed.

Update Workstation

Purpose: Displays all pending text or graphics on the workstation.

Format: Update Workstation
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

–1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine has different effects on different types of devices:

- **Printers.** All pending graphics are displayed and paper is advanced to top of form.
- **Plotters.** Has no effect on plotters.
- **CRT Screens.** Has no effect on CRT screens.

Paging Routines

This section describes the VDI routines that copy, write, and determine the aspects of the active or nonactive pages. The paging routines apply only to CRT device drivers.

If the video adapter does not support separate buffer areas for cursor text and graphics, data in one mode is destroyed when exiting to the other mode. When entering cursor text mode, the active cursor text page is cleared to blank spaces. When entering graphics mode, the active graphics page is cleared to the workstation's background color.

Whether the active cursor page is cleared, or all cursor pages are cleared when entering cursor text mode is device-dependent. It is also device-dependent whether the active graphics page is cleared or all graphics pages are cleared when entering graphics mode. To ensure that all pages are cleared upon changing the active page, issue a Clear Workstation command.

Routines included in this section are:

- Copy Page
- Inquire Page
- Set Page.

Copy Page

Purpose: Allows the application to copy the contents of one page (active or nonactive) into another page.

Format: Copy Page**
(handle, source, destination)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

source
The source page number to be copied.

destination
The destination page number.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

–1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Note: If the Inquire Error routine returns an error code of –3095, an invalid page number was specified in *source* or *destination.* No copy was performed.

Remarks: The copy is done on pages in the current displaying mode. If the display is in cursor text mode, cursor text pages are copied; if the display is in graphics mode, graphics pages are copied. If either or both of the pages specified in *source* and *destination* are not available on a device, no copy is done.

Inquire Page

Purpose: Allows the application to determine which page is the active page, the number of available pages, the existence of paging capability, and the visual page (page being displayed).

Format: Inquire Page**
(handle, graphics mode, cursor mode)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **graphics mode**
A three-element array containing information about the graphics mode pages.

graphics mode(1)
Graphics mode page currently selected for display.

graphics mode(2)
Graphics mode page currently selected for reading and writing.

graphics mode(3)
Number of graphics mode pages available (numbered 0 through *graphics mode(3)* minus 1).

cursor mode
A three-element array containing information about the cursor mode pages.

cursor mode(1)
Cursor text mode page currently selected for display. The default is page 0.

cursor mode(2)
Cursor text mode page currently selected for reading and writing.

cursor mode(3)
Number of cursor text mode pages available (numbered 0 through *cursor mode(3)* minus 1).

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The number of pages available in cursor text mode can vary depending upon the mode. If the display is in cursor text mode, the number of pages available corresponds to the current cursor text mode. If the display is in graphics mode, the number of pages available corresponds to the mode that takes effect on the next Enter Cursor Addressing Mode routine (see "Enter Cursor Addressing Mode").

Set Page

Purpose: Allows the application to write to the specified page (may be the same page as visual page).

Format: Set Page**
(handle, graphics mode in, cursor mode in, graphics mode out, cursor mode out)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

graphics mode in
A two-element array of the graphics mode page requested.

> **graphics mode in(1)**
> Graphics mode page requested for display.
>
> **graphics mode in(2)**
> Graphics mode page requested for reading and writing.

cursor mode in
A two-element array of the cursor text mode page requested.

> **cursor mode in(1)**
> Cursor text mode page requested for display. Default is page 0.
>
> **cursor mode in(2)**
> Cursor text mode page requested for reading and writing. Default is page 0.

Output:

graphics mode out
A two-element array containing information about graphics mode pages.

> **graphics mode out(1)**
> Graphics mode page currently selected for display.
>
> **graphics mode out(2)**
> Graphics mode page currently selected for reading and writing.

cursor mode out
A two-element array containing information about cursor text mode pages.

cursor mode out(1)
Cursor text mode page currently selected for display.

cursor mode out(2)
Cursor text mode page currently selected for reading and writing.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

–1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The Open Workstation routine sets both pages to 0. The current cursor text mode referenced above is the current mode if the display is in cursor text mode, not that which takes effect at the next Enter Cursor Addressing Mode routine.

Note: Valid page numbers for both graphics mode and cursor mode are determined by the Inquire Page routine. If an invalid page number is requested for the Set Page routine, the corresponding current page does not change. Since no error is indicated for an invalid request, you may want to include a comparison of the *graphics/cursor mode in* parameter to the *graphics/cursor mode out* parameter, within your application program.

Pel Routines

This section describes the VDI routines and their parameters used to move one or more pels (pixels) on the display screen.

There are two methods of moving pels:

- Copying pels from one location on the display screen to another location on the display screen.
- Getting pels from the current page, putting them in temporary storage (a user defined array), and then moving them to the display screen (which may or may not be a new page).

Routines included in this section are:

- Copy Pels
- Get Pels
- Put Pels.

Copy Pels

Purpose: Copies one or more pels from one position (source rectangle) to another position (destination or target rectangle) on the same display page. This routine works in the graphics mode.

Format: Copy Pels**
(handle, xy)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

xy
A six-element array containing the diagonal coordinates of the source rectangle (which contains the pels) on the display screen and the coordinates of the lower left-hand corner of the destination rectangle on the same display screen.

xy(1)
The x-coordinate of the lower left-hand corner of the source rectangle in NDC units.

xy(2)
The y-coordinate of the lower left-hand corner of the source rectangle in NDC units.

xy(3)
The x-coordinate of the upper right-hand corner of the source rectangle in NDC units.

xy(4)
The y-coordinate of the upper right-hand corner of the source rectangle in NDC units.

xy(5)
The x-coordinate of the lower left-hand corner of the target (destination) rectangle in NDC units.

xy(6)
The y-coordinate of the lower left-hand corner of the target rectangle in NDC units.

Output: None.

Status: The status is a value the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The target or destination rectangle is clipped as necessary at the top and right edges of the screen. The writing modes in Pel routines operate on the hardware (bit map) color indexes, not VDI color indexes.

The writing of the pels at the target rectangle is performed according to the current writing mode as set by the Set Writing Mode routine (see "Set Writing Mode").

Get Pels

Purpose: Moves the pels from the display page into an array where they are stored in anticipation of moving them to the display (which may or may not be a new page). This routine works in the graphics mode.

Format: Get Pels**
(handle, xy, destination)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

xy
A four-element array containing the diagonal coordinates of the rectangle containing the pels.

> **xy(1)**
> The x-coordinate of the lower left-hand corner of the rectangle in NDC units.
>
> **xy(2)**
> The y-coordinate of the lower left-hand corner of the rectangle in NDC units.
>
> **xy(3)**
> The x-coordinate of the upper right-hand corner of the rectangle in NDC units.
>
> **xy(4)**
> The y-coordinate of the upper right-hand corner of the rectangle in NDC units.

Output:

destination
Name of the array where you are going to store the pels until you move them to the display screen.

Status: The status is a value that the routine returns to indicate its successful completion.

〉0=Indicates the number of 16-bit words in *source* required to hold the pels.

0=No error.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see “Inquire Error”).

Remarks: The following C language example computes the number of bytes required in the array for the Get Pels routine:

```
#include <stdio.h>

main()
{
/*******************************************************/
/*                                                     */
/*      Compute the number of bytes required           */
/*      in the array for the Get Pels routine.         */
/*                                                     */
/*******************************************************/
short paspect;
char response;

long bytes, xdll, ydll, xdur, ydur, xpix, ypix, xll, yll;
long xur, yur, xmul, ymul, xsurf, ysurf, xmax, ymax;
int bits, planes, screen, legal_coords;

static struct {
     long  x_dev;
     long  y_dev;
     long  xpix_size;
     long  ypix_size;
     long  bits_pixel;
     long  clr_planes;
} screen_info[] = {

{720, 512, 285, 285, 1, 1}, /* vdiamg display */
{720, 512, 285, 285, 1, 4}, /* vdiacg display*/
{1024,768, 285, 285, 1, 1}, /* vdiemg display*/
{640, 350, 328, 508, 1, 2}, /* vdiega display, 4 color*/
{640, 350, 328, 508, 1, 4}/* vdiega display, 16 color*/

};
```

```
printf(
"This Program computes the size required (in bytes)\n\r"
);
printf(
"for the array into which the Get Pels routine\n\r"
);
printf(
"puts the pel image from the screen\n\r\n\r"
);

/*

        find out whether preserve aspect ratio mode is used
*/

printf(
"Workstation opened with preserved aspect ratio? (Y/N):"
);
scanf("%c%*c", &response);
response |= 0x20; /* ignore case */
while (!((response == 'n') || ( response == 'y'))){
   printf("Please answer Y or N: ");
   scanf("%c%*c", &response);
   response |= 0x20; /* ignore case */
}

if (response == 'y')
   paspect = 1;
else
   paspect = 0;

/*

       find out screen in use

*/
```

```
do{

   printf(
   "\n\r1) vdiamg\n\r2) vdiacg\n\r3) vdiemg\n\r"
   );
   printf(
   "4) vdiega 4 color\n\r5) vdiega 16 color\n\r\n\r"
   );
   printf(
   "Please enter one of the selections shown: "
   );
   scanf("%d%*c", &screen);

   if((screen < 1) || (screen > 5))
      printf("Incorrect selection\n\r");

} while((screen < 1) || (screen > 5));

screen--; /* compute the info table index */
/*

   compute tranform multiplier and
   maximum 32k space values

*/
xmul = screen_info[screen].x_dev;
ymul = screen_info[screen].y_dev;

if( paspect ){
   /* X surface size in microns */
   xsurf = screen_info[screen].x_dev *
           screen_info[screen].xpix_size;
   /* Y surface size in microns */
   ysurf = screen_info[screen].y_dev *
           screen_info[screen].ypix_size;

   if ( xsurf <= ysurf )
      xmul = ysurf / screen_info[screen].xpix_size ;
   else
      ymul = xsurf / screen_info[screen].ypix_size ;
}
```

```
/*

        compute maximum 32k space x and y

*/
xmax = ((screen_info[screen].x_dev * 32768)/ xmul ) - 1;
ymax = ((screen_info[screen].y_dev * 32768)/ ymul ) - 1;

/*

         get rectangle corners

*/
do{
   do{
      legal_coords = 1; /* assume legal coords. */
      printf(
      "Enter coordinates of lower left corner (x,y): "
      );
      scanf("%ld %ld%*c", &xll, &yll);
      if((xll < 0) || (xll > xmax)){
         legal_coords = 0;
         printf("\n\rX ranges from 0..%d\n\r", xmax);
      }
      if((yll < 0) || (yll > ymax)){
         legal_coords = 0;
         printf("\n\rY ranges from 0..%d\n\r", ymax);
      }
   }while(!legal_coords);

   do{
      legal_coords = 1; /* assume legal coords. */
      printf(
      "Enter coordinates of upper right corner (x,y): "
      );
      scanf("%ld %ld%*c", &xur, &yur);
```

```
            if((xur < 0) || (xur > xmax)){
               legal_coords = 0;
               printf("\n\rX ranges from 0..%d\n\r", xmax);
            }
            if((yur < 0) || (yur > ymax)){
               legal_coords = 0;
               printf("\n\rY ranges from 0..%d\n\r", ymax);
            }
         }while(!legal_coords);

         if ((xll > xur) || (yll > yur)){
            legal_coords = 0;
            printf("\n\rLower left corner must be");
            printf("below upper right corner.");
            printf(" Please renter both.\n\r");
         }
      }while( !legal_coords );

      /*

                Transform rectangle corners.
                The coordinates, (xdll,ydll) and (xdur,ydur),
                are pixel-space corners.

      */
      xdll = (int)((long)(xll * xmul) / 32768);
      ydll = (int)((long)(yll * ymul) / 32768);
      xdur = (int)((long)(xur * xmul) / 32768);
      ydur = (int)((long)(yur * ymul) / 32768);
      /*

                compute bytes required in array

      */
```

```
bits = screen_info[screen].bits_pixel;
planes = screen_info[screen].clr_planes;
xpix = xdur - xdll + 1;
ypix = ydur - ydll + 1;
bytes = (int)((long)(xpix * bits + 7)/8);
bytes = 4 + (bytes * planes * ypix);

printf("%ld bytes are required to ", bytes);
printf("hold your pixels.\n\r");

}
```

Put Pels

Purpose: Displays pels that were stored with the Get Pels routine on the current page. This routine works in the graphics mode.

Format: Put Pels**
(handle, xy, source)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

xy
A two-element array containing the coordinates (on the display screen) of the target or destination rectangle.

xy(1)
The x-coordinate of the lower left-hand corner of the target rectangle in NDC units.

xy(2)
The y-coordinate of the lower left-hand corner of the target rectangle in NDC units.

source
The name of the array used in a Get Pels routine to store the pels prior to moving them (see "Get Pels").

Output: None.

Status: The status is a value the routine returns to indicate its successful completion.

0 = No error.

-1 = An error occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The destination rectangle is clipped as necessary at the top and right edges of the screen. Writing modes in Pel routines operate on the hardware (bit map) color indexes, not on the VDI color indexes.

Cursor Control Routines

Cursor Control routines are used to position the standard alphanumeric cursor, as well as cursor text on the display screen. This is important to applications that present a fill-in-the-form operation.

You can position the cursor on a display screen cell grid of rows and columns. A typical display screen measures 24 rows by 80 columns. The position (row 1, column 1) in the top left corner of the display screen is the home position.

The cursor routines affect only CRT devices. The CRT devices open in the default graphics mode. You must call the Enter Cursor Addressing Mode routine to use other cursor routines.

All routines in this section are escape routines. In using them, your application program becomes device-dependent.

Routines included in this section are:

- Cursor Down
- Cursor Home
- Cursor Left
- Cursor Right
- Cursor Up
- Direct Cursor Address
- Enter Cursor Addressing Mode
- Erase to End of Line
- Erase to End of Screen
- Exit Cursor Addressing Mode
- Inquire Addressable Character Cells
- Inquire Current Cursor Text Address
- Inquire Cursor Text Mode
- Output Cursor Addressable Text
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index
- Set Cursor Text Mode.

Cursor Down

Purpose: The cursor moves down one row on the cursor cell grid without moving horizontally, when in cursor mode.

Format: Cursor Down**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices. As the cursor moves down, displayed text is scrolled upward after the bottom of the screen is reached.

Cursor Home

Purpose: The cursor moves to the upper left corner (row 1, column 1) of the display screen, when in the cursor mode.

Format: Cursor Home**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices. No action occurs if the cursor is already in the home position.

Cursor Left

Purpose: The cursor moves to the left one column without changing its vertical position, when in cursor mode.

Format: Cursor Left**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices. No action occurs if the cursor is already in column 1.

Cursor Right

Purpose: The cursor moves right one column without changing its vertical position, when in cursor mode.

Format: Cursor Right**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices. No action occurs if the cursor is already in the far right column.

Cursor Up

Purpose: The cursor moves up one row on the cursor cell grid without moving horizontally, when in cursor mode.

Format: Cursor Up**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices. No action occurs if the cursor is already on the top row.

Direct Cursor Address

Purpose: The cursor moves to a specified position on the display screen.

Format: Direct Cursor Address**
(handle, row, column)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

row
The row number of a new cursor location.

column
The column number of a new cursor location.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices. Specify the position by *row* and *column*. If you specify a position off the screen, the cursor moves to the closest possible position.

Enter Cursor Addressing Mode

Purpose: Prepares the display screen for future cursor-addressing routines and places the cursor at the top left position on the display screen, or "home."

Format: Enter Cursor Addressing Mode**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine is applicable only to CRT devices. Use this routine to exit the graphics mode and enter the cursor mode.

The display screen is cleared when the graphics mode is exited.

You must use this routine before you can use any of the routines affecting the cursor.

When entering cursor mode, all cursor text pages are cleared to blank spaces.

Erase to End of Line

Purpose: Erases all text from the current cursor-address to the end of the line (row). The cursor remains at the current address when the routine is complete.

Format: Erase to End Of Line**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

–1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices.

Erase to End of Screen

Purpose: Erases all text from the cursor position to the end of the display screen. No action occurs if no text extends below or to the right of the cursor.

Format: Erase to End of Screen**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

–1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices. The routine erases all the lines from the cursor position to the end of the screen. If the cursor is in the upper left position of the screen, the routine erases the entire screen. If the cursor is in a central position on the screen, the routine erases the rest of the line the cursor is on, and all following lines to the end of the screen.

The cursor remains at the current address upon completion of this routine.

Exit Cursor Addressing Mode

Purpose: Exits the cursor mode if not in graphics mode.

Format: Exit Cursor Addressing Mode**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: To do graphics after entering cursor mode, you must use this routine to enter the graphics mode from the cursor-addressing mode. It is necessary to be in graphics mode for any graphics, alpha text, or graphics text processing operation.

The graphics mode is the open workstation default condition.

When exiting a cursor page, the new graphics page is cleared to the current background color.

Inquire Addressable Character Cells

Purpose:	Returns the number of addressable columns and rows.
Format:	Inquire Addressable Character Cells (handle, rows, columns)
Input:	**handle** The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.
Output:	**rows** Number of addressable rows (–1 means cursor-addressing not possible).
	columns Number of addressable columns (–1 means cursor-addressing not possible).
Status:	The status is a value that the routine returns to indicate its successful completion.
	0 = No error.
	–1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").
Remarks:	None.

Inquire Current Cursor Text Address

Purpose: Returns the current cursor position to the application program.

Format: Inquire Current Cursor Text Address**
(handle, row, column)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **row**
Row number of the current cursor position.

column
Column number of the current cursor position.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices.

Note: Since error -3095 will be returned if this routine is called while the cursor is in graphics mode, this routine can be used to determine the current mode (cursor text or graphics).

Inquire Cursor Text Mode

Purpose: Returns the mode the workstation is put in when the next Enter Cursor Addressing Mode routine call is received.

Format: Inquire Cursor Text Mode**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = Currently selected mode.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine is not supported by any of the device drivers in the IBM RT PC Graphics Development Toolkit.

Output Cursor Addressable Text

Purpose: Writes text at the current cursor position, and then moves the cursor one space to the right for each character in the text string.

Format: Output Cursor Addressable Text**
(handle, char string)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

char string
A string of characters to be displayed on the output device.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: To use this routine, you must be in cursor mode. The text is placed on the cursor-addressable cell grid (usually 24 rows by 80 columns). New text writes over the text in the same location.

You can place only one line of text with this routine if the device does not have character wraparound. To begin a new line, use the Direct Cursor Address routine to place the cursor on the next line at the far left column (see "Direct Cursor Address"). The Output Cursor Addressable Text routine can then be repeated.

The CRT devices supported by the IBM RT PC Graphics Development Toolkit do not have character wraparound.

Reverse Video Off

Purpose: Puts foreground color into the foreground and background color into the background.

Format: Reverse Video Off**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The routine affects only cursor text. This routine is ignored if reverse video is already off.

Reverse Video On

Purpose: Puts foreground color into the background and background color into the foreground.

Format: Reverse Video On**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The routine affects only cursor text. Note that this routine is ignored if reverse video is already on.

Set Cursor Text Attributes

Purpose: Sets the attributes of blink, bold, reverse video, and underline for subsequent cursor-addressable text.

Format: Set Cursor Text Attributes**
(handle, req att, sel att)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

req att
A four-element array that contains the requested cursor text attributes.

req att(1)
Requested reverse video mode:

0=Disable reverse video
1=Enable reverse video
2=Do not change current state
3=Toggle reverse video attributes

req att(2)
Requested underline cursor text mode:

0=Disable underline cursor text
1=Enable underline cursor text
2=Do not change current state
3=Toggle underline cursor text attributes

req att(3)
Requested blink text mode:

0=Disable blink cursor text
1=Enable blink cursor text
2=Do not change current state
3=Toggle blink cursor text attributes

req att(4)
Requested bold cursor text mode:

0 = Disable bold cursor text
1 = Enable bold cursor text
2 = Do not change current state
3 = Toggle bold cursor text attributes

Output: **sel att**
A four-element array that contains the selected cursor text attributes.

sel att(1)
Selected reverse video mode:

0 = Disabled
1 = Enabled

sel att(2)
Selected underline cursor text mode:

0 = Disabled
1 = Enabled

sel att(3)
Selected blink cursor text mode:

0 = Disabled
1 = Enabled

sel att(4)
Selected bold cursor text mode:

0 = Disabled
1 = Enabled

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine can set reverse video mode, underline cursor text mode, blink cursor text mode, and bold cursor text mode, or inquire about these attributes.

Note: You can use this routine to do an inquiry of the current attributes by setting all modes to not change the current state (2). The current state is then returned in the *sel att* array.

Set Cursor Text Color Index

Purpose: Sets the foreground and background colors for the cursor-addressable text.

Format: Set Cursor Text Color Index**
(handle, fore requested, back requested, fore selected, back selected)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

fore requested
The color index of the foreground of subsequent output cursor text (default 1).

back requested
The color index of the background of subsequent output cursor text (default 0).

Output:

fore selected
Color index selected for cursor text foreground.

back selected
Color index selected for cursor text background.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

−1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: If an invalid color index is selected, the closest valid color index is chosen. The background color (cursor cell background) is changed immediately.

Set Cursor Text Mode

Purpose: Sets the mode to be used when the next Enter Cursor Addressing Mode routine call is received.

Format: Set Cursor Text Mode**
(handle, mode requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

mode requested
The type of screen mode that is requested.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = mode requested.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine is not supported by any of the device drivers in the IBM RT PC Graphics Development Toolkit.

General Graphics Routines

This group of graphics routines is general in nature. They involve the setting of color modes, writing modes, and displaying or removing the graphics cursor. You must be in graphics mode, and not cursor mode, to use these routines.

General Graphics Routines included in this part of the Graphics Routines section are:

- Display Graphic Input Cursor
- Inquire Color Representation
- Inquire Graphic Color Burst Mode
- Remove Graphic Input Cursor
- Set Background Color Index
- Set Color Representation
- Set Graphic Color Burst Mode
- Set Writing Mode.

Display Graphic Input Cursor

Purpose: Displays a graphics input cursor at a specified location on the workstation.

Format: Display Graphic Input Cursor**
(handle, x, y)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

x
x-coordinate of new cursor location in NDC units.

y
y-coordinate of new cursor location in NDC units.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine works only on CRT devices. To use this routine you must be in the graphics mode.

The graphics cursor is the same as the one used for feedback by the Input Locator routine (for example, crosshairs or arrow). The Input Locator routine automatically displays a cursor when it needs one (see "Input Locator (request mode)"). You do not have to reference this routine when using Input Locator.

Inquire Color Representation

Purpose: Returns information about the color table.

Format: Inquire Color Representation
(handle, index requested, set flag, rgb)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
The color index you are asking about (0 to a device maximum).

set flag
If you make this flag=0, the RGB color intensities for the color index requested are returned in the *rgb* array.

If you make this flag=1, the RGB color intensities of the actual color index used by the device driver are returned in the *rgb* array. These realized intensities reflect the color index chosen by the driver if a color is not supported by the device.

Output:

rgb
A three-element array of color intensities (in tenths of a percent, 0-1000).

rgb(1)
Red intensity, (in tenths of a percent, 0-1000).

rgb(2)
Green intensity, (in tenths of a percent, 0-1000).

rgb(3)
Blue intensity, (in tenths of a percent, 0-1000).

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Color index selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Use this routine to ask what intensities of red, green, and blue are associated with a color index (see "Set Color Representation"). You have a choice to ask about the intensities in the table or the actual intensities realized on a device.

Inquiring about a color not offered by a device returns values for the closest index in the table. The values chosen will be available if *set flag* = 1. Inquiring about a color index not found in the table returns values for the color index closest to the requested index.

Inquire Graphic Color Burst Mode

Purpose: Returns the mode that the workstation (when the workstation is in graphics mode) is going to use when the next Exit Cursor Addressing Mode routine is used.

Format: Inquire Graphic Color Burst Mode**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Mode selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine is not supported by any of the device drivers in the IBM RT PC Graphics Development Toolkit.

Remove Graphic Input Cursor

Purpose: Removes the graphics input cursor from its current location on the workstation.

Format: Remove Graphic Input Cursor**
(handle)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine affects only CRT devices. Use this routine to remove the graphics input cursor displayed by Display Graphic Input Cursor (see "Display Graphic Input Cursor"). You do not have to reference this routine when performing Locator Input.

Set Background Color Index

Purpose: Sets the background color on the workstation.

Format: Set Background Color Index
(handle, index requested)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
A color index selected from the color table.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Color index selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The color change will not appear until the next Clear Workstation routine is called (see "Clear Workstation"). If the color index is not valid, the background color does not change.

Background color cannot be set on printers and plotters.

Set Color Representation

Purpose: Changes the color representation of a color index.

Format: Set Color Representation
(handle, index requested, rgb requested, rgb realized)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
The color index you select from the color table (0 to device maximum).

rgb requested
A three-element array of color intensities (in tenths of a percent, 0-1000).

rgb requested(1)
Red intensity, (in tenths of a percent, 0-1000).

rgb requested(2)
Green intensity, (in tenths of a percent, 0-1000).

rgb requested(3)
Blue intensity, (in tenths of a percent, 0-1000).

Output:

rgb realized
A three-element array of color intensities selected (in tenths of a percent, 0-1000).

rgb realized(1)
Red intensity, (in tenths of a percent, 0-1000).

rgb realized(2)
Green intensity, (in tenths of a percent, 0-1000).

rgb realized(3)
Blue intensity, (in tenths of a percent, 0-1000).

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Index selected.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Use this routine to change or add colors to the default color table. See Figure 3-2.

The maximum number of colors in a color representation table is the number of colors that a workstation can display at one time.

The number of colors that can be displayed on a workstation simultaneously is returned in the 14th element of the Open Workstation *workout* array (see "Open Workstation"). A zero is returned in the 36th element of the same *workout* array, if the device is not capable of color.

Color Table					
Color Index	Color	Red	Green	Blue	Binary Number
0	Black	0	0	0	0000
1	White	1000	1000	1000	0001
2	Red	1000	0	0	0010
3	Green	0	1000	0	0011
4	Blue	0	0	1000	0100
5	Yellow	1000	1000	0	0101
6	Cyan	0	1000	1000	0110
7	Magenta	1000	0	1000	0111
〉7	White	1000	1000	1000	xxxx

Figure 3-2. Color Table

Note: All intensities in the table are in tenths of a percent.

The maximum color index allowed is one less than the maximum number of colors in the color table, because the color index starts at zero and continues to n - 1 (the *workout(40)* array contains the number of colors allowed).

You can select individual colors if the device allows individual color selections. With a palette oriented device, palettes are selected.

Changing a Color

You can change a color in the table by changing the percentages of red, green, and blue intensities associated with a color index. To do this, assign new values to the *rgb requested* array in this routine.

The percentages are expressed in tenths of a percent. Multiply the percent value you intend to use by 10 and use that value in the *rgb requested* array. For example, to get a display of 75% of the total red color available in the pels on a device, use a value of 750 for red in the *rgb requested* array.

The following example is a segment of C code that sets the color representation for color indexes 8-15 on the IBM Enhanced Graphics Adapter device driver. When this device driver is first opened, indexes 8-15 are defined as white, or RGB values of (1000, 1000, 1000). In the device driver for the IBM Enhanced Graphics Adapter, each RGB component can support four levels: 0, 333, 666, and 1000. The example defines color indexes 8-15 as one more level of blue than its corresponding element for the indexes 0-7.

```
#define RED 0
#define GREEN 1
#define BLUE 2

   struct{
      short requested[3];
      short selected[3];
   } rgb;

   short device_handle,
         i,
         set_flag,
         req_index,
         sel_index;
```

```
/*

        set index 0 to light grey

*/
   req_index = 0;
   rgb.requested[RED] = 666;
   rgb.requested[GREEN] = 666;
   rgb.requested[BLUE] = 666;

   sel_index = vs_color( device_handle,
                         req_index,
                         rgb.requested,
                         rgb.selected );

/*

         set color index one to light green

*/
   req_index = 1;
   rgb.requested[RED] = 333;
   rgb.requested[GREEN] = 1000;
   rgb.requested[BLUE] = 333;

   sel_index = vs_color( device_handle,
                         req_index,
                         rgb.requested,
                         rgb.selected );

/*

         set realized flag to 1

*/
   set_flag = 1;

   for ( req_index = 7 ; req_index < 16 ; req_index++){

/*

         inquire color representation of index i

*/
```

```
          sel_index = vq_color( device_handle,
                                req_index,
                                set_flag,
                                rgb.selected );


          if( rgb.selected[BLUE] < 1000){
             rgb.requested[BLUE] += 333;
/*

             set new color representation

*/
             sel_index = vs_color( device_handle,
                                   req_index,
                                   rgb.requested,
                                   rgb.selected   );

          }
       }
```

Set Graphic Color Burst Mode

Purpose: Sets the mode that the workstation uses when the next Exit Cursor Addressing Mode routine is received.

Format: Set Graphic Color Burst Mode**
(handle, mode)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

mode
Condition of color burst:

0=Color burst off
1=Color burst on

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0=No error.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine is not supported by any of the device drivers in the IBM RT PC Graphics Development Toolkit.

Set Writing Mode

Purpose: Controls how output writes over pending graphics in a print buffer or on a display screen.

Format: Set Writing Mode
(handle, mode requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

mode requested
This is a number you choose from the Boolean Operation Chart located in this routine under "Remarks:."

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Mode selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Set Writing Mode allows you to mix colors. When one area (source) is moved or written to overlap another area (destination) a new color is formed at the destination.

You can calculate the new color of the destination by performing a Boolean operation on the source and destination binary color numbers. See Figures 3-3 and 3-4. The binary color numbers are shown in the Set Color Representation routine (see "Set Color Representation").

Convert the regular color index from the base 10 to the base 2, to get binary numbers not listed in the color table.

Figure 3-3 shows the Boolean operation associated with each mode number. In the table, D is the binary color number of the destination area to be overlapped. S is the binary color number of the source area to be moved or written to the destination. If you are unfamiliar with Boolean algebra, consult a computer math book for more information.

Mode	Boolean Operation
1	D=0 (all color bits off)
2	D=(D AND S)
3	D=(NOT D) AND S
4	D=S (replace)
5	D=D AND (NOT S)
6	D=D
7	D=D XOR S
8	D=D OR S (overstrike)
9	D=NOT (D OR S)
10	D=NOT (D XOR S)
11	D=NOT D
12	D=(NOT D) OR S
13	D=NOT S
14	D=D OR (NOT S)
15	D=NOT (D AND S)
16	D=1 (white)

Figure 3-3. Boolean Operation Chart

The following table shows the color result of some operations. The binary color number is 0010 for red and 0011 for green.

Mode	Operation	Destination (D)	Source (S)	Result
1	D=0	0010 (2) (RED)	0011 (3) (GREEN)	0000 (0) (BLACK)
2	D=D AND S	0010 (2) (RED)	0011 (3) (GREEN)	0010 (2) (RED)
4	D=S	0010 (2) (RED)	0011 (3) (GREEN)	0011 (3) (GREEN)
6	D=D	0010 (2) (RED)	0011 (3) (GREEN)	0010 (2) (RED)
7	D=D XOR S	0010 (2) (RED)	0011 (3) (GREEN)	0001 (1) (WHITE)
8	D=D OR S	0000 (0) (BLACK)	0001 (1) (WHITE)	0001 (1) (WHITE)

Figure 3-4. Writing Mode Color Results

Remarks: The default writing mode is 4 when a workstation is opened. When a Boolean operation results in a color index that has no assignment in the color table, the destination becomes white.

Graphics Primitives

The Graphics Primitives routines are used to output lines, markers, text strings, and other graphics primitives. Note that you must be in Graphics mode, and not Cursor mode, to use these routines. The following list shows the Graphics Primitives Routines:

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Inquire Cell Array

Purpose: Returns the color indexes used to produce a specific cell array.

Format: Inquire Cell Array
(handle, xy, row length, number rows, elements per row, rows used, value flag, colors)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

xy
A four-element array that contains the lower left corner and upper right corner coordinates of the cell array.

xy(1)
x-coordinate of the lower left corner.

xy(2)
y-coordinate of the lower left corner.

xy(3)
x-coordinate of the upper right corner.

xy(4)
y-coordinate of the upper right corner.

row length
Length of each row in the *colors* array.

number rows
Number of rows in the *colors* array.

Output:

elements per row
Number of elements actually used in *row length*.

rows used
Number of rows used in the *colors* array.

value flag
This flag is set to 0 if there are no errors. It is set to 1 if a color value could not be determined for some pel.

colors
Color indexes for each cell are returned to this array. The size of the array needs to be *(number rows* multiplied by *row length)* long.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine returns color indexes, one row at a time, starting from the top left corner of the rectangular area, proceeding downward. See "Output Cell Array" for information regarding how the rectangular area is divided.

Inquire Current Fill Area Attributes

Purpose: Returns the current fill area attributes for polygons, circles, pie slices, and bars.

Format: Inquire Current Fill Area Attributes
(handle, attributes)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **attributes**
A four-element array indicating current fill area attributes.

attributes(1)
Interior fill style, (1 to 3, see "Set Fill Interior Style").

attributes(2)
Current fill area color index.

attributes(3)
Current fill area style index, (1 to device maximum, see Set Fill Style Index routine).

attributes(4)
Current writing mode.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Inquire Current Polyline Attributes

Purpose: Returns all current polyline features that affect lines, arcs, and current writing mode to your application program.

Format: Inquire Current Polyline Attributes
(handle, attributes)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **attributes**
A four-element array containing the attributes of the current line such as type, color, writing mode, width.

attributes(1)
Current polyline type index, (1 to device maximum, see "Set Polyline Line Type" routine).

attributes(2)
Current polyline color index.

attributes(3)
Current writing mode (see "Set Writing Mode" routine).

attributes(4)
Current line width in NDC units.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Inquire Current Polymarker Attributes

Purpose: Returns the current polymarker features to your application program.

Format: Inquire Current Polymarker Attributes
(handle, attributes)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **attributes**
A four-element array containing the current marker characteristics.

attributes(1)
Current polymarker type, (1 to device maximum, see "Set Polymarker Type" routine).

attributes(2)
Current polymarker color index.

attributes(3)
Current writing mode (see "Set Writing Mode" routine).

attributes(4)
Current polymarker height in NDC units.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None

Output Arc

Purpose: Draws an arc on a workstation.

Format: Output Arc
(handle, x, y, radius, start angle, end angle)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

x
x-coordinate of the arc center point in NDC units.

y
y-coordinate of the arc center point in NDC units.

radius
The length of the radius of the arc, measured on the x axis in NDC units.

start angle
Starting angle of the arc in tenths of degrees, (0-3600).

end angle
Ending angle of the arc in tenths of degrees, (0-3600).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine draws arcs using current polyline attributes. Define an arc by giving its center point, two end points and the radius. The radius is measured along the x-axis (horizontal). The start and end angles are measured on the circle where zero degrees is located 90 degrees (at 3 o'clock) to the right of vertical. Degree values increase in a counterclockwise direction.

For an arc, the radius specified is assumed to be along the x-axis, and takes priority over the radius that is determined by the center point and an arbitrary point of the arc. For an example of arc, refer to "Graphics Primitive Output Hints" in Chapter 2.

Output Bar

Purpose: Produces a rectangular area on a workstation using fill area attributes.

Format: Output Bar
(handle, xy)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

xy
A four-element array containing the coordinates of the lower left and the upper right corners of the bar.

xy(1)
x-coordinate of the lower left corner in NDC units.

xy(2)
y-coordinate of the lower left corner in NDC units.

xy(3)
x-coordinate of the upper right corner in NDC units.

xy(4)
y-coordinate of the upper right corner in NDC units.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Use this routine to draw a rectangular area on a workstation. The bar will have the current features of fill area color, interior type, and fill style. "Hollow" bars are outlined with a solid border.

Output Cell Array

Purpose: Produces a rectangular or square area divided into color cells of equal size and shape.

Format: Output Cell Array
(handle, xy, row length, elements per row, number rows, writing mode, colors)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

xy
A four-element array containing the x and y coordinates of the lower left and upper right corner of a rectangle or square.

> **xy(1)**
> x-coordinate of the lower left corner in NDC units.
>
> **xy(2)**
> y-coordinate of the lower left corner in NDC units.
>
> **xy(3)**
> x-coordinate of the upper right corner in NDC units.
>
> **xy(4)**
> y-coordinate of the upper right corner in NDC units.

row length
Length of each row in the *colors* array.

elements per row
Number of elements actually used in *row length.*

number rows
Number of rows in the *colors* array.

writing mode
Pel operation to be performed.

colors
An array of color indexes you choose from the color table. The size of the array needs to be *number rows* times *row length* (rows multiplied by columns) long.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: A cell array is a rectangular or square area divided into color cells of equal size and shape. Each cell can display an individual color (any valid color defined in the color table).

The color index for each cell is set in the *colors* array. The number of elements in this array is the product of *row length* and *number rows.* Color indexes are stored by rows, and each row has a length of *elements per row* (number of columns).

When the Output Cell Array routine is called, the number of rows displayed in the cell array is *number rows.* The number of columns displayed is *elements per row.*

Some of the color indexes stored in the *colors* array are not used when displaying the cell array. The number of color indexes not used is the difference between *row length* and *elements per row.* For example, if a *colors* array was defined to have a *row length* of five, three *elements per row,* and one *number rows,* the cell array would have three columns by one row. The color indexes used to display the three cells are: *colors(1), colors(2),* and *colors(3).*

With a *colors* array defined to have a *row length* of five, three *elements per row,* and two *number rows,* the cell array would have three columns by two rows. The color indexes used to display the six cells are: *colors(1), colors(2),* and *colors(3)* in row one, and *colors(6), colors(7),* and *colors(8)* in row two for the respective columns.

If the workstation is capable of the Output Cell Array routine, it is returned in the 39th element of the Open Workstation *workout* array (see "Open Workstation").

Output Circle

Purpose: Draws a circle on a workstation using fill area attributes.

Format: Output Circle
(handle, x, y, radius)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

x
x-coordinate of circle center in NDC units.

y
y-coordinate of circle center in NDC units.

radius
The radius of the circle as measured on the x-axis in NDC units.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Use this routine to draw a circle on a workstation. The circle will have the currently selected features of fill area color, fill style, and fill interior style.

The circle is specified by length of radius and the x and y coordinates of the center of the circle. The length of the radius is measured as if it were laid out on the x-axis.

For a circle, the radius specified is assumed to be along the x-axis and takes priority over the radius that is determined by the center point and an arbitrary point of the circle. For an example of circle, refer to "Graphics Primitive Output Hints" in Chapter 2.

Output Filled Area

Purpose: Produces a filled area on a workstation.

Format: Output Filled Area
(handle, count, xy)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

count
Number of vertexes in polygon.

xy
An array of the x and y coordinates of the corners of the polygon. The number of elements in this array must be twice the value of *count.*

xy(1)
x-coordinate of first polygon corner in NDC units.

xy(2)
y-coordinate of first polygon corner in NDC units.

xy(3)
x-coordinate of second polygon corner in NDC units.

xy(4)
y-coordinate of second polygon corner in NDC units.

xy(n–1)
x-coordinate of *n*/2 corner of polygon in NDC units.

xy(n)
y-coordinate of *n*/2 corner of polygon in NDC units.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

–1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Use this routine to draw a polygon on a graphics device. The polygon will have corners at the points you determine in the coordinate array. The polygon will have the current features of fill style, fill color, and fill interior style. Hollow filled polygons are outlined with a solid border.

Make sure the *count* and the size of the *xy* coordinate array correspond. The number of elements in this array must be twice the value of *count*. For example, to output a fill area with 6 points, you need a *count* of 6 and an array size of 12.

Output Pie Slice

Purpose: Draws a pie slice on a workstation using fill area attributes.

Format: Output Pie Slice
(handle, x, y, radius, start angle, end angle)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

x
x-coordinate of pie slice center point in NDC units.

y
y-coordinate of pie slice center point in NDC units.

radius
The radius of the pie slice as measured on the x-axis in NDC units.

start angle
The place on the circle where the pie slice is to start in tenths of degrees (0-3600).

end angle
The place on the circle where the pie slice is to end in tenths of degrees (0-3600).

Output: None.
Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Use this routine to draw a pie slice on a workstation. The pie slice is drawn counterclockwise from the *start angle* to the *end angle*. The pie slice is affected by the current features of fill color, fill style, and fill interior style.

Output Pie Slice

A pie slice is specified by the coordinates of the center point, the radius, and the start and end angles. The radius is measured along the x- (horizontal) axis. The start and end angles are measured on the circle where zero degrees is located 90 degrees (at 3 o'clock) to the right of vertical. Degree values increase in a counterclockwise direction.

For a pie slice, the radius specified is assumed to be along the x-axis and takes priority over the radius that is determined by the center point and an arbitrary point of the arc.

Output Polyline

Purpose: Draws one or more line segments on a workstation.

Format: Output Polyline
(handle, count, xy)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

count
Number of points through which lines are to be drawn.

xy
An array of coordinates of the line to be drawn. The number of elements in this array must be twice the value of *count.*

> **xy(1)**
> x-coordinate of the starting point in NDC units.
>
> **xy(2)**
> y-coordinate of the starting point in NDC units.
>
> **xy(3)**
> x-coordinate of the second point in NDC units.
>
> **xy(4)**
> y-coordinate of the second point in NDC units.
>
> **xy(5)**
> x-coordinate of the third point in NDC units.
>
> **xy(6)**
> y-coordinate of the third point in NDC units.
>
> **xy(n–1)**
> x-coordinate of the *n*/2 point in NDC units.
>
> **xy(n)**
> y-coordinate of the *n*/2 point in NDC units.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine draws a line that begins with coordinates of the first position in the array, and passes through the positions of each successive set of coordinates in the array.

Make sure the *count* and the size of the *xy* coordinate array correspond. The number of elements in this array must be twice the value of *count*. For example, to output polylines connecting 6 points, you need a *count* of 6 and an array size of 12.

This routine draws polylines using current polyline attributes.

Output Polymarker

Purpose: Displays markers on a workstation.

Format: Output Polymarker
(handle, count, xy)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

count
This is the number of markers to be placed on the device.

xy
An array containing the x and y coordinates of each marker. The number of elements in this array must be twice the value of *count.*

> **xy(1)**
> x-coordinate of first marker in NDC units.
>
> **xy(2)**
> y-coordinate of first marker in NDC units.
>
> **xy(3)**
> x-coordinate of second marker in NDC units.
>
> **xy(4)**
> y-coordinate of second marker in NDC units.
>
> **xy(n–1)**
> x-coordinate of *n*/2 marker in NDC units.
>
> **xy(n)**
> y-coordinate of *n*/2 marker in NDC units.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine draws polymarkers using current polymarker attributes. It is important that you ensure that the *count* and the size of the *xy* coordinate array correspond. The number of elements in this array must be twice the value of *count*. For example, to output 6 polymarkers, you need a *count* of 6 and an array size of 12.

Set Fill Color Index

Purpose: Sets the color of subsequent filled areas. These include fill areas, circles, pie slices, and bars.

Format: Set Fill Color Index
(handle, index requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
The fill color index you request (0 to device maximum).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Color index selected.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: At least two colors are provided, foreground and background. The color index ranges from 0 to some device-dependent maximum. If the color specified is invalid, the closest value within the range is selected.

Set Fill Interior Style

Purpose: Sets the style of interior for subsequent circles, polygons, pie slices, and bars.

Format: Set Fill Interior Style
(handle, style requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

style requested
One of four styles requested. The styles available are:

0=Hollow
1=Solid
2=Pattern
3=Hatch

Note: The pattern and hatch styles are the same if the device does not support native patterns.

Output: None.
Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Style selected.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Choose from four styles of interiors. Use the number that is associated with a particular style for the *style requested* variable in the routine call. The capability of a device to fill a polygon is returned in the 38th element of the Open Workstation *workout* array (see "Open Workstation"). The number of patterns offered by a device is returned in the 12th element. The number of hatch styles offered by a device is returned in the 13th element.

Set Fill Style Index

Purpose: Selects a fill style based on the fill interior style.

Format: Set Fill Style Index
(handle, index requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
Requested fill style index for pattern or hatch fill.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Index selected.

−1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Set Fill Style Index

Remarks: The index references a hatch style if the fill interior style is Hatch, or a pattern if the fill interior style is Pattern. At least six hatch styles are available. The hatch styles are defined in Figure 3-5.

Hatch Styles	
Style Number	Description
1	Narrow spaced 45 degree lines.
2	Medium spaced 45 degree lines.
3	Wide spaced 45 degree lines.
4	Narrow spaced 45 degree lines crossed with –45 degree lines.
5	Medium spaced 45 degree lines crossed with –45 degree lines.
6	Wide spaced 45 degree lines crossed with –45 degree lines.

Figure 3-5. Hatch Styles

More than six patterns may be available from a particular device. If you ask for a style that is out of range, style index 1 is selected by the device driver.

The number of hatch styles available is returned in the 13th element of the Open Workstation *workout* array (see “Open Workstation”).

Set Polyline Color Index

Purpose: Sets the color of all lines and arcs that follow this routine.

Format: Set Polyline Color Index
(handle, index requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
The polyline color index you want to use (0 to device maximum).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Color index selected.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine sets the color index in which subsequent lines are displayed. At least two colors are provided, foreground and background. Color indexes range from 0 to a device-dependent maximum. If the color specified is invalid, the closest value in range is chosen.

Set Polyline Line Type

Purpose: Sets the line style of subsequent lines and arcs.

Format: Set Polyline Line Type
(handle, type requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

type requested
A line type chosen from 1 to some device-dependent maximum.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Line type selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The number of line types available is returned in the 7th element of the Open Workstation *workout* array (see "Open Workstation"). If you request a line type that is out of range, the workstation uses line style number 1 (solid style).

Set Polyline Line Width

Purpose: Sets the width for all subsequent lines and arcs.

Format: Set Polyline Line Width
(handle, width requested)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

width requested
The line width you want in NDC units.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Line width selected.

−1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The default is a line width of one pel. The number of line widths available on a device is returned in the 8th element of the Open Workstation *workout* array (see "Open Workstation"). The minimum and maximum line widths are returned in the 63rd and 64th elements, respectively.

Set Polymarker Color Index

Purpose: Sets the color of markers to be drawn on the workstation.

Format: Set Polymarker Color Index
(handle, index requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
An index you select from the color table (0 to device maximum).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Color index selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Use this routine to set the color of markers to be drawn by the Polymarkers routine. Choose a color index as described in "Set Color Representation."

The polymarker color is affected by the current writing mode. If you are using a mode other than replace, the color you get may be different from the color you request.

Color indexes range from 0 to a device-dependent maximum. If the color specified is invalid, the closest value in range is chosen.

Set Polymarker Height

Purpose: Sets the height of subsequent markers displayed on a workstation.

Format: Set Polymarker Height
(handle, height requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

height requested
Requested marker height in NDC units.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Height selected.

−1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Use this routine to set the size of subsequent markers displayed on a workstation. The smallest size allowed for a particular device is returned in the 65th element of the Open Workstation *workout* array (see "Open Workstation"). The largest size is returned in the 66th element. The number of distinct sizes available is returned in the 10th element of that same *workout* array.

If you request a size that is outside the capabilities of the device, the closest available size is returned. If the requested size is not an exact size offered by a device, the next smallest size is returned.

Set Polymarker Type

Purpose: Selects the type of marker to be displayed on the workstation.

Format: Set Polymarker Type
(handle, type requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

type requested
Type of marker chosen from the following marker types:

1 = Dot
2 = Cross
3 = Star
4 = Square
5 = X
6 = Diamond
⟩6 = Device-dependent

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0 = Marker type selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Six marker types are guaranteed. Use the numbers for the *type requested* parameter in the routine call. More than six marker types may be available. The number of types available is device-dependent and is returned in the 9th element of the Open Workstation *workout* array (see "Open Workstation"). If the requested type is out of range, type 3 is used.

Graphics Text Routines

This group of VDI routines determines the type of text to be used. The routines set the height, alignment, font, and other text conditions. Note that you must be in Graphics mode, and not Cursor mode, to use these routines.

Graphics Text Routines included in this part of the Graphics Routines section are:

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Inquire Current Graphic Text Attributes

Purpose: Returns all current attributes that affect graphic text, such as text size, text color, text font, and text rotation.

Format: Inquire Current Graphic Text Attributes
(handle, attributes)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **attributes**
A 10-element array containing information about current graphics text characteristics.

attributes(1)
Current graphics text font.

attributes(2)
Current graphics text color index.

attributes(3)
Current angle of rotation of text baseline (in tenths of degrees 0-3600).

attributes(4)
Current horizontal alignment:

0=Left justified (default)
1=Center justified
2=Right justified

attributes(5)
Current vertical alignment:

0=Bottom justified (default)
1=Center justified
2=Top justified

attributes(6)
Current writing mode. See Boolean operation chart in the Set Writing Mode routine.

attributes(7)
Current character width in NDC units.

attributes(8)
Current character height in NDC units.

attributes(9)
Current character cell width in NDC units.

attributes(10)
Current character cell height in NDC units.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Output Graphic Text

Purpose: Writes graphics text on a workstation.

Format: Output Graphic Text
(handle, x, y, char string)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

x
x-coordinate of alignment point of text in NDC units.

y
y-coordinate of alignment point of text in NDC units.

char string
A string of characters to be displayed on the output device.

Note: Characters must be printable characters (between ASCII 032 and ASCII 126). All others are ignored.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The text is aligned to the position you specify in the parameters in the routine call, and has the current selections for color, font, height, alignment, and rotation (see "Set Graphic Text Alignment").

Any graphics text character outside the actual device coordinates is not displayed.

Set Character Height

Purpose: Sets the height of graphics text.

Format: Set Character Height
(handle, height requested, char width, cell width, cell height)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

height requested
The character height you requested in NDC units.

Output:

char width
Actual character width used by the driver in NDC units.

cell width
Character cell width in NDC units.

cell height
Character cell height in NDC units.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Height selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The height of the text is defined as the height of the tallest (baseline to top) character in the font. You select the height in NDC units; VDI selects the width of the characters, and the cell width and height. This maintains the correct proportions for the graphics font.

The next smaller size is used when the requested size is not offered by a device. The default size is one that permits a device to display 24 characters vertically and 80 characters horizontally. The number of text sizes available on a device is returned in the 6th element of the Open Workstation *workout* array (see "Open Workstation").

Set Graphic Text Alignment

Purpose: Sets graphics text horizontal and vertical alignment.

Format: Set Graphic Text Alignment
(handle, horizontal requested, vertical requested, horizontal realized, vertical realized)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

horizontal requested
Horizontal alignment requested (applies to the character body width not the cell width):

0=Left justified (default)
1=Center justified
2=Right justified

Note: If an invalid horizontal alignment is requested, the *horizontal requested* parameter is reset to its default.

vertical requested
Vertical alignment requested:

0=Bottom justified (default)
1=Center justified
2=Top justified

Note: If an invalid vertical alignment is requested, the *vertical requested* parameter is reset to its default.

Output:

horizontal realized
Horizontal alignment selected by the driver.

vertical realized
Vertical alignment selected by the driver.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The default alignment places the bottom left corner of the first character (not the character cell) in the string at the graphics text position.

Set Graphic Text Color Index

Purpose: Sets the graphics text color index.

Format: Set Graphic Text Color Index
(handle, index requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
The color index you request (0 to device maximum).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Color index selected.

–1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Color indexes range from 0 to a device-dependent maximum. If a color index requested is not valid, the closest color index in range is selected. However, the routine always returns the color index selected.

Set Graphic Text Font

Purpose: Selects the hardware text font for graphics text.

Format: Set Graphic Text Font
(handle, font requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

font requested
Requested hardware graphics text font number.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Font selected.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Availability of graphics text fonts is device-dependent. At least one is available on any device. If you request a font outside the capability of the device, font number one is selected.

The number of graphics text fonts available on a device is returned in the 11th element of the Open Workstation *workout* array (see "Open Workstation").

Set Graphic Text String Baseline Rotation

Purpose: Sets the baseline rotation of a string of graphics text characters.

Format: Set Graphic Text String Baseline Rotation
(handle, angle requested)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

angle requested
Requested angle of rotation of the character string baseline, in tenths of degrees (0-3600).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Angle selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The entire string (rather than the individual character) is rotated to the angle you specify in the *angle requested* parameter. Angles are determined on the unit circle, and increase in a counterclockwise direction. When the angle requested is out of range, a character baseline of 0 degrees is used. When text is rotated some of it may disappear from the workstation.

The ability of a device to rotate text is returned in the 37th element of the Open Workstation *workout* array (see "Open Workstation").

Alpha Text Routines

The Toolkit includes many VDI routines that control the placement of document quality text on a workstation. This type of quality text is available in graphics mode, and is called alpha text.

You can control multiple fonts, interline spacing, underlining, placement, superscripting, subscripting, and other features by using alpha text routines.

Alpha text is displayed to the best resolution and accuracy of the workstation. It is defined in NDC units. All numeric sizes and locations are in NDC units.

Routines included in this section are:

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript/Superscript Mode
- Set Alpha Text Underline Mode.

Inquire Alpha Text Capabilities

Purpose: Returns information regarding the alpha text features of the workstation.

Format: Inquire Alpha Text Capabilities
(handle, capabilities)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **capabilities**
A 15-element array containing returned information regarding the alpha text capabilities of the device.

capabilities(1)
Superscript capability:

0=No
1=Yes

capabilities(2)
Subscript capability:

0=No
1=Yes

capabilities(3)
Underline capability:

0=No
1=Yes

capabilities(4)
Overstrike capability:

0=No
1=Yes

capabilities(5)
Number of discrete alpha text sizes (1 to device maximum).

capabilities(6)
Discrete size of the default font.

capabilities(7)
Character positioning capability flag:

0=Characters positioned on cell boundaries.

1=Characters positioned on a finer grid than a character cell, not necessarily the same grid as graphics.

capabilities(8)
The number of horizontal character cell positions across the workstation in the default font. For a typical display or printer that can place text only on cell boundaries, this is 80.

capabilities(9)
The number of vertical character cell positions down the workstation in the default font. This is 24 for a typical display and 66 for a typical printer that can place text only on cell boundaries.

capabilities(10)
The number of horizontal character cell positions represented by the distance specified in *capabilities(14)*. Use *capabilities(14)* divided by *capabilities(10)* to determine the width of a character cell.

capabilities(11)
The number of vertical character cell positions represented by the distance specified in *capabilities(15)*. Use *capabilities(15)* divided by *capabilities(11)* to determine the height of a character cell.

capabilities(12)
The number of horizontal alpha text grids represented by the distance specified in *capabilities(14)*. Use *capabilities(14)* divided by *capabilities(12)* to determine the width of an alpha text grid.

capabilities(13)
The number of vertical alpha text grids represented by the distance specified in *capabilities(15)*. Use *capabilities(15)* divided by *capabilities(13)* to determine the height of an alpha text grid.

capabilities(14)
The width in NDC units of the number of character cells (in the default font) specified in *capabilities(10)*. Use *capabilities(14)* divided by *capabilities(10)* to determine the width of a character cell.

capabilities(15)
The height in NDC units of the number of character cells (in the default font) specified in *capabilities(11)*. Use *capabilities(15)* divided by *capabilities(11)* to determine the height of a character cell.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Inquire Alpha Text Cell Location

Purpose: Returns the location of an alpha text character cell to your application program.

Format: Inquire Alpha Text Cell Location
(handle, row, column, prop flag, xout, yout)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

row
Row number of character cell.

column
Column number of character cell.

Output:

prop flag
Proportional spacing flag:

0 = No proportional spacing
1 = Proportional spacing

If this value is 1, then the position represented by the x and y coordinates may not be accurate.

xout
x-coordinate of lower left corner of character cell in NDC units.

yout
y-coordinate of lower left corner of character cell in NDC units.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Inquire Alpha Text Font Capability

Purpose: Returns features of a particular alpha text font and size.

Format: Inquire Alpha Text Font Capability
(handle, font requested, size requested, capabilities)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

font requested
A font index.

size requested
Size of text in the desired font ranging from 1 to a device maximum.

Output:

capabilities
A seven-element array containing returned information regarding the alpha text font capabilities.

capabilities(1)
The number of default horizontal character cells across the display surface.

0=Requested font not available
-1=Proportional font

capabilities(2)
The number of vertical character cells down the workstation in the font (0=font not available).

capabilities(3)
The number of horizontal character cell positions represented by the distance specified in *capabilities(6)*. Use *capabilities(6)* divided by *capabilities(3)* to determine the width of a character cell (0=font not available).

capabilities(4)
The number of vertical character cell positions represented by the distance specified in *capabilities(7)*. Use *capabilities(7)* divided by *capabilities(4)* to determine the height of the character cell.

capabilities(5)
Proportional spacing flag:

0 = No
1 = Yes

capabilities(6)
The width, in NDC units, of the number of character cells (in the selected font) specified in *capabilities(3)*. Use *capabilities(6)* divided by *capabilities(3)* to determine the width of a character cell, including any round off error. This value is not accurate if the proportional spacing flag is set to 1, since the character cell size is not constant (0 = font not available).

capabilities(7)
The height, in NDC units, of the number of character cells (in the selected font) specified in *capabilities(4)*. Use *capabilities(7)* divided by *capabilities(4)* to determine the height of a character cell (0 = font not available).

Status: The status is a value that the routine returns to indicate its successful completion.

0 = Not available.

>0 = Font availability.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see “Inquire Error”).

Remarks: None.

Inquire Alpha Text Position

Purpose: Returns the current alpha text position to your application program.

Format: Inquire Alpha Text Position
(handle, x out, y out)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **x out**
x-coordinate of text position in NDC units.

y out
y-coordinate of text position in NDC units.

Status: The status is a value that the routine returns to indicate its successful completion.

0=No error.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The alpha text position is returned in NDC units (from 0 to device maximum). The *workout(52)* parameter contains the maximum NDC units for the x-axis and the *workout(53)* parameter contains the maximum NDC units for the y-axis. The position of (0,0) is the lower left corner of the display.

Inquire Alpha Text String Length

Purpose: Returns the length of the text string, based on the current font in use.

Format: Inquire Alpha Text String Length
(handle, char string)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

char string
A string of characters to be displayed on the output device.

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Length of text string in NDC units.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: If a control character (ASCII values 0-31 or 〉126) appears in the string, it terminates the string, and the string length up to that point is returned. This routine is useful when using proportional fonts, since each character is not the same width. It is also useful for adjusting the space between words, since multiplication of the width of a character cell in NDC units may produce inaccurate results due to the inherent round off error in the character cell size reported back to your program.

Output Alpha Text

Purpose: Writes the text string at the current alpha text position.

Format: Output Alpha Text
(handle, char string, x out, y out)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

char string
A string of characters to be displayed on the output device.

Output: **x out**
The x-coordinate of the text position after the text string has been written. This is the same value that would be returned if Inquire Alpha Text Position were invoked (see "Inquire Alpha Text Position").

y out
The y-coordinate of the text position after the text string has been written. This is the same value that would be returned if Inquire Alpha Text Position were invoked (see "Inquire Alpha Text Position").

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: All current alpha text features are honored. The first text position must be defined in the Set Alpha Text Position routine (see "Set Alpha Text Position"). This routine changes the alpha text position to the end of the text string after it writes the text.

Placement of the carriage return ASCII character in the string causes the alpha text position to be set to the beginning of the line. Placement of a line feed control character causes the alpha text position to be advanced by the current line spacing.

All other control characters (ASCII values 0-31 or >126) are not written. Attempting to display characters in a position past the x or y maximum of the display produces device-dependent results.

Set Alpha Text Color Index

Purpose: Selects the color of subsequent alpha text.

Format: Set Alpha Text Color Index
(handle, index requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

index requested
The color index you select from the color table (0 to a device maximum).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0 = Color index selected.

−1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Color indexes range from 0 to a device-dependent maximum. If the color specified is invalid, the closest value in range is chosen.

Set Alpha Text Font and Size

Purpose: Sets the hardware alpha text font and size for subsequent writing of alpha text.

Format: Set Alpha Text Font and Size
(handle, font requested, size requested, capabilities)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

font requested
A number from the list below. If the desired font is not available, the standard font (1), is used:

1=`Normal/standard font (the default)`
2=**`Bold--always provided for printers`**
3=*`Italics`*
4=Proportionally spaced normal font
5=**Proportionally spaced bold**
6=*Proportionally spaced italics*
>6=Device-dependent

size requested
Size of the text wanted. The size number ranges from 1 to a device maximum. See Appendix C for device-dependent information.

Output:

capabilities
An eight-element array containing returned information regarding the alpha text font capabilities of the device.

capabilities(1)
Font size selected.

capabilities(2)
The number of horizontal character cell positions across the workstation in this font. This is −1 if a proportional font is selected, since the character cell size is not constant.

capabilities(3)
Default number of vertical character cells down the workstation in this font.

capabilities(4)
Number of horizontal character cell positions represented by the distance specified in *capabilities(7)*. Use *capabilities(7)* divided by *capabilities(4)* to determine the width of a character cell.

capabilities(5)
Number of vertical character cell positions represented by the distance specified in *capabilities(8)*. Use *capabilities(8)* divided by *capabilities(5)* to determine the height of a character cell.

capabilities(6)
Proportional spacing flag:

0 = No
1 = Yes

If this value is 1, then the size represented by *capabilities(7)* and *capabilities(8)* may not represent the selected font.

capabilities(7)
The width in NDC units of the number of character cells (in the selected font) specified in *capabilities(4)* Use *capabilities(7)* divided by *capabilities(4)* to determine the width of a character cell (not accurate with proportional spacing).

capabilities(8)
The height in NDC units of the number of character cells (in the selected font) specified in *capabilities(5)*. Use *capabilities(8)* divided by *capabilities(5)* to determine the height of character cell.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = Font unavailable

⟩0 = Font available

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Set Alpha Text Line Spacing

Purpose: Sets the vertical spacing between lines of alpha text.

Format: Set Alpha Text Line Spacing
(handle, spacing requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

spacing requested
Line spacing requested (a positive value in NDC units).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Line spacing selected in NDC units.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Vertical spacing is the amount of movement down the page when a line feed control character is received in a string of alpha text. The default is the amount of space between the lines of alpha text equal to the default character cell height.

Line spacing must always be a positive value. It changes the y-coordinate of the alpha text position when a line feed is encountered. You need to update the line spacing to the character cell height of a new font whenever fonts are changed.

Set Alpha Text Overstrike Mode

Purpose: Turns overstriking on or off.

Format: Set Alpha Text Overstrike Mode
(handle, mode requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

mode requested
Overstrike mode wanted:

0 = Overstrike mode off (default)
1 = Overstrike mode on

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Mode selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: The default is overstriking off. When overstriking is on, the alpha text position is not automatically advanced after each character. You must change the position using the Set Alpha Text Position routine (see "Set Alpha Text Position"). Carriage return and line feed characters can still modify the current alpha text position.

Set Alpha Text Pass Through Mode

Purpose: Turns pass-through mode on or off.

Format: Set Alpha Text Pass Through Mode
(handle, mode requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

mode requested
Pass-through mode wanted:

0 = Pass-through mode off (default)
1 = Pass-through mode on

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Mode selected.

−1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: Pass-through mode enables you to pass all characters to a workstation using the Output Alpha Text routine (see "Output Alpha Text"). Control characters are ignored, except in pass-through mode wherein they are sent to the workstation.

When this mode is in effect, the alpha text position is not automatically updated. All alpha text features may not be honored. This routine can be used to send device-dependent setup strings to a particular device.

Set Alpha Text Position

Purpose: Sets the position of a string of alpha text characters on the workstation.

Format: Set Alpha Text Position
(handle, x in, y in, x out, y out)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

x in
The x-coordinate of the lower left corner of the alpha text string location selected in NDC units.

y in
The y-coordinate of the lower left corner of the alpha text string location selected in NDC units.

Output:

x out
Actual x-coordinate of text position location that was selected by the driver in NDC units.

y out
Actual y-coordinate of text position location that was selected by the driver in NDC units.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine only sets the position of the text. Use the Output Alpha Text routine to write it (see "Output Alpha Text"). The lower left corner of the first character (not the character cell) in the string will be at the alpha text position.

Set Alpha Text Quality

Purpose: Sets the alpha text quality level.

Format: Set Alpha Text Quality
(handle, mode requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

mode requested
Text quality requested (0-100%) (Default=100, highest quality).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0=Mode selected.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: If an invalid mode is requested, the default (100) is selected. In draft quality mode (0), small imperfections due to bidirectional printing or print head speeds are accepted. In high quality range, the output is the highest quality possible using the device hardware.

Set Alpha Text Subscript/Superscript Mode

Purpose: Causes alpha text to be offset below or above the text line.

Format: Set Alpha Text Subscript/Superscript Mode
(handle, mode requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

mode requested
A mode number from the following list:

0 = Subscripting and superscripting off (default)
1 = Subscripting on
2 = Superscripting on

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥0 = Mode selected.

−1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see “Inquire Error”).

Remarks: None.

Set Alpha Text Underline Mode

Purpose: Turns alpha text underlining on or off.

Format: Set Alpha Text Underline Mode
(handle, mode requested)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

mode requested
Underline mode requested:

0 = Underlining off (default)
1 = Underlining on

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

≥ 0 = Mode selected.

-1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Input Routines

This section describes the VDI input routines that supply graphic information to the application program.

Input routines operate in two modes, request and sample. In request mode, the input device is activated and waits for the application user to present a device-specific signal when the data is ready to be entered. The signal may be pressing the enter key, a function key, or a switch to return the value. In the sample mode, the pending input is returned to the application program immediately. There are four types of graphics input, as follows:

- **Choice.** The choice input routines return the choice selection that has been pressed.
- **Locator.** The locator input routines return to the application program the point coordinates in NDC units of the locator device. A locator device may be a mouse, crosshair, joystick, trackball, or set of cursor keys.
- **String.** String input routines return text strings entered on the keyboard.
- **Valuator.** The Valuator input routines return a scalar value between 0 and 32767 corresponding to the status code of a valuator device. A valuator device may be a potentiometer or slide control.

Routines included in this section are:

- Input Choice (request mode)
- Input Choice (sample mode)
- Input Locator (request mode)
- Input Locator (sample mode)
- Input String (request mode)
- Input String (sample mode)
- Input Valuator (request mode)
- Input Valuator (sample mode)
- Read Cursor Movement Keys
- Set Line Edit Characters.

Input Choice (request mode)

Purpose: Activates the choice device (the most common are function keys) and waits for a selection before returning to the application program.

Format: Input Choice
(handle, initial choice, final choice)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

initial choice
Any choice number from 1 to a device-dependent maximum (see Open Workstation). Use this in conjunction with the status code to evaluate the final choice.

Output: **final choice**
A choice number returned to the program indicating which function key was pressed by the application user.

Status: The status is a value that the routine returns to indicate its successful completion.

0=Request unsuccessful, but no error occurred.

〉0=Request successful.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see “Inquire Error”).

Remarks: None.

Input Choice (sample mode)

Purpose: Polls the choice device. If a choice is pending, it is returned.

Format: Input Choice
(handle, final choice)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **final choice**
The choice number is returned to this variable when the sample is taken. A zero is returned if a choice has not been made at the time of the sample.

Status: The status is a value that the routine returns to indicate its successful completion.

0=Sample unsuccessful, but no error occurred.

)0=Sample successful.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Input Locator (request mode)

Purpose: Causes the graphics cursor to be displayed on a workstation until a selection is made.

Format: Input Locator
(handle, initial xy, ink, rubberband, echo handle, final xy, terminator)

Input:

handle
The unique device ID returned by the Open Workstation routine for the input device when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

initial xy
A two-element array containing the initial locator coordinates.

initial xy(1)
x-coordinate of locator initial position in NDC units.

initial xy(2)
y-coordinate of locator initial position in NDC units.

ink
The method of movement of the cursor is device-dependent. If *ink* is on, a line is drawn between the initial locator position and the final locator position. The line has the current polyline attributes, such as color and line type. The inking mode choices available are:

0=Off
1=On

rubberband
If a rubberband line is chosen, a movable line is drawn between the initial locator position and the current position of the locator device.

The line changes dynamically as the input device changes position. When the locator is terminated, the last rubberband line is removed from the display surface. See "Remarks:" for more information on rubberbanding. The rubberbanding mode numbers are:

0=Rubberbanding off
1=Rubberband line
2=Rubberband rectangle

echo handle
Device ID for the output device that displays the tracking cross (graphics cursor). This ID is returned when the workstation is opened. The *handle* and the *echo handle* may be the same, indicating that there is only one device for both input and output.

Output:

final xy
A two-element array containing the final locator coordinates.

final xy(1)
x-coordinate of locator final position in NDC units.

final xy(2)
y-coordinate of locator final position in NDC units.

terminator
An ASCII value from the input device is returned to this variable. For keyboard terminated locator input, this is the ASCII value of the key struck to terminate input. For non-keyboard terminated input (for example, tablet or mouse) the valid locator terminators begin with ASCII value 032 (space) and increase from there.

Status:

The status is a value that the routine returns to indicate its successful completion.

0=Request unsuccessful.

>0=Request successful.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Note: If this routine returns an error code of -2692, the specified *echo handle* is not valid.

Remarks:

If rubberband rectangle is specified, a rectangle is drawn using the initial locator position as one corner, and the current position of the locator device as the opposite corner. The rectangle changes dynamically as the input device changes position. When the locator is terminated, the last rubberband rectangle is removed from the workstation. If an invalid mode is specified, then rubberbanding is turned off.

Note: The rubberbanding function is device-dependent and may not be available with all devices supported by this routine.

When locator is invoked, a tracking cross appears on the screen at the initial locator position. The cross is moved with some graphics input device (such as a mouse) or by pressing one of the cursor movement keys (up, down, left, right).

Initially, the graphics cursor moves in large increments. Pressing the **Insert** key decreases the size of the movements for cursor keys. Pressing the **Insert** key again, causes the increments of movement to toggle back to the previous size.

When the cross is at a desired location, the point is entered by pressing either a button on the graphics input device or by pressing any alpha key. When a point is entered, its coordinates are returned to the application program.

Input Locator (sample mode)

Purpose: Returns the current position of the graphics input cursor without waiting for operator interaction.

Format: Input Locator
(handle, xyin, xyout, pressed, released, keystate)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

xyin
The initial locator position in NDC units.

xyin(1)
Initial x position of locator.

xyin(2)
Initial y position of locator.

Output:

xyout
The current locator position in NDC units.

xyout(1)
Current x position of locator.

xyout(2)
Current y position of locator.

pressed
An integer that represents those buttons that have changed state from *released* to *pressed* since the last input request.

released
An integer representing those buttons that have changed state from *pressed* to *release* since the last input request.

keystate
This is the current button state for the input device.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = Sample unsuccessful.

≥ 1 = Sample successful.

−1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: None.

Input String (request mode)

Purpose: Accepts character input from the keyboard and waits for the input before proceeding with the application.

Format: Input String
(handle, max length, echo mode, echo xy, char string)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

max length
Maximum string length.

echo mode
An integer indicating whether the string will be displayed or not (cannot be used in cursor mode):

0 = Do not display input characters
1 = Display input characters

echo xy
A two-element array containing the position of the characters if they are displayed.

echo xy(1)
x-coordinate of the text display position in NDC units.

echo xy(2)
y-coordinate of the text display position in NDC units.

Output:

char string
Output string passed from the keyboard to the application program. The *char string* will be less than or equal to the *max length* parameter.

Status: The status is a value that the routine returns to indicate its successful completion.

0=Request unsuccessful.

〉0=Request successful.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine activates the keyboard and any characters up to a Return terminator or line feed are returned. The Return terminator or line feed is not included in the characters returned.

Line editing characters have their normal effect and can be used if errors are made. For the default editing characters, see "Set Line Edit Characters." The maximum string length must be ≥ 1. This routine terminates when the maximum length is reached or a line terminator has been entered.

Input String (sample mode)

Purpose: Polls the keyboard of the system.

Format: Input string
(handle, max length, echo mode, echo xy, char string)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

max length
Maximum string length.

Note: *char string* is the length of the string returned or the length of *max length* (whichever is smaller).

echo mode
An integer indicating whether the string will be displayed or not (cannot be used in cursor mode):

0=Do not display input characters
1=Display input characters

echo xy
A two-element array containing the position of the characters if they are displayed.

> **echo xy(1)**
> x-coordinate of the text display position in NDC units.
>
> **echo xy(2)**
> y-coordinate of the text display position in NDC units.

Output:

char string
Output string passed from the keyboard to the application program. *char string* is less than or equal to the *max length* parameter.

Input String (sample mode)

Status: The status is a value that the routine returns to indicate its successful completion.

0=Sample unsuccessful (characters not available).

⟩0=Sample successful (value returned equals number of characters returned).

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see “Inquire Error”).

Remarks: The line editing characters affect the input to this routine. For information on editing characters, see “Set Line Edit Characters.” If there is any pending input, it is returned. The input is stopped when the queue is empty, if a line terminator is encountered, or if the maximum string length is exceeded. See Appendix C, “Graphics Drivers”, for information on the string device for the workstation.

Input Valuator (request mode)

Purpose: Activates the valuator (potentiometer) device and the user sets it to the desired value.

Format: Input Valuator
(handle, initial value, echo handle, final value)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

initial value
An initial value for valuator to start from, in NDC units. When the valuator device is moved, the final value is calculated from this.

echo handle
Device ID for the output device that displays the tracking cross (graphics cursor). This ID is returned when the workstation is opened.

Output:

final value
The value on the device after this routine has run. This value is returned in a 0-32767 range to indicate the valuator position.

Status: The status is a value that the routine returns to indicate its successful completion.

0=Request unsuccessful.

〉0=Request successful.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine is not supported by any of the device drivers in the IBM RT PC Graphics Development Toolkit.

Input Valuator (sample mode)

Purpose: Returns the current value of the valuator device without waiting for operator interaction.

Format: Input Valuator
(handle, final value)

Input: **handle**
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

Output: **final value**
The current valuator value if the sample was successful. This value is returned in a 0-32767 range to indicate the valuator position.

Status: The status is a value that the routine returns to indicate its successful completion.

0=Sample unsuccessful.

>0=Sample successful.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine is not supported by any of the device drivers in the IBM RT PC Graphics Development Toolkit.

Read Cursor Movement Keys

Purpose: Determines if a cursor movement key was struck and returns the resultant direction in integer form.

Format: Read Cursor Movement Keys**
(handle, input mode, direction, key)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

input mode
Input mode:

1=Request
2=Sample

Output:

direction
A number indicating the direction key pressed is returned to this variable. The following list describes the values returned to the direction variable:

-1=No keystroke occurred (sample mode only)
0=A keystroke besides a cursor movement key was struck
1=Down and left
2=Down
3=Down and right
4=Left
5=Value not defined
6=Right
7=Up and left
8=Up
9=Up and right

key
If a cursor movement key was struck, the ASCII decimal equivalent value of the key (1 to 9) is returned to this variable.

Status: The status is a value that the routine returns to indicate its successful completion.

0 = No error.

–1 = An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine can be used in either the graphics or cursor mode.

Set Line Edit Characters

Purpose: Sets the current line editing character.

Format: Set Line Edit Characters
(handle, line del, char del)

Input:

handle
The unique device ID returned by the Open Workstation routine when the workstation is opened. Refers to a specific graphics device when multiple workstations are open.

line del
Character to use to delete the current line (the **Ctrl_U** or ASCII NAK (ASCII value 21) is the default).

char del
Character to use to delete previous character (the **Ctrl_H** or ASCII Backspace (ASCII value 8) is the default).

Output: None.

Status: The status is a value that the routine returns to indicate its successful completion.

0=No error.

-1=An error has occurred. The actual error can be retrieved by invoking the Inquire Error routine (see "Inquire Error").

Remarks: This routine applies to input string routines only.

Error Handling

This section describes the error handling feature when the routines return an error code. The Inquire Error routine is the only Toolkit routine call described in this section.

Inquire Error

Purpose:	Returns the last error that has occurred.
Format:	Inquire Error
Input:	None.
Output:	None.
Status:	The status is a value that the routine returns to indicate the type of error that has occurred.

≥ 0 = No error.

<0 = The error code. Refer to Appendix D, “Error Codes”, for an explanation of each error code.

Remarks: This routine returns the actual last error encountered. You call this routine after another VDI routine returns a −1 status.

Note: Refer to Appendix D, “Error Codes”, for information about error handling and for the cause of error conditions.

Appendix A. Installing the Graphics Development Toolkit

The Graphics Development Toolkit is a library of software routines that enables you to develop device-independent application programs. This appendix explains how to install the IBM RT PC Graphics Development Toolkit product. Before you install the Toolkit on your system, the AIX Operating System and the Toolkit device drivers must be installed. These device drivers are provided as part of the IBM RT PC Multi-User programs.

The **/etc/master** file of the AIX Operating System contains the default size for shared memory areas. This default is 512K bytes. The Graphics Development Toolkit requires that this size remain at least 512K bytes.

Installation Procedure

This section describes the installation procedure. If any error messages occur during the procedure, see the *IBM RT PC Messages Reference.*

The Graphics Development Toolkit product includes a single diskette. Remove this diskette from the plastic envelope at the back of the binder.

1. Make sure that no one else is using the system and that no user programs are running. If the system is not in a quiet state, problems may occur as you install the various files for your licensed product.

2. Log in as super-user or as a member of the system group. You must have super-user authority, or be a member of the system group, to install a licensed program product. See the *IBM RT PC Using and Managing the AIX Operating System* for more information.

   After you log in, you will see the # prompt.

3. Type **installp** then press **Enter.**

4. The following message appears to remind you to make sure that the system is quiet:

   ```
   000-123 Before you continue, you must make sure there
           is no other activity on the system. You
           should have just restarted the system, and no
           other users should be logged on. Refer to
           your messages reference book for more
           information.

           Do you want to continue with this command?
           (y or n)
   ```

   Type **y** and press **Enter** to continue with the installp command.

5. Insert the program diskette in response to the prompt. Then press **Enter.**

   ```
   Insert the program diskette into diskette drive
   "/dev/rfd0" and then press Enter.
   ```

6. In response to the prompt, type **y** to indicate that you wish to continue with the installation. Then, press **Enter.**

   ```
   The program "Graphics Development Toolkit"
   will be installed.

   Do you want to do this? (y/n)
   ```

7. If a current version of this program has already been installed on your system, a message explains that the version of the program you are about to install is the same as or older than the version you already have installed on your system. Indicate whether you wish to go ahead with the installation.

   ```
   You are about to install version "xx.xx.xxxx" of this
   program. This version is the same as or older than
   the version currently on your system.
   Do you want to do this? (y/n)
   ```

   If you type **y** and press **Enter**, the installation process begins.

   ```
   Please insert the diskette in /dev/rfd0.
   ```

Your program diskette should already be in diskette drive /dev/rfd0. **Type Return** is the same as **Press Enter.** As installation continues, various files are listed on the screen as they are copied to the fixed disk.

8. When installation is complete, remove the program diskette from the diskette drive and replace it in its protective envelope in the binder.

   ```
   The installation process has completed.
   ```

9. Log off as super-user or as a member of the system group.

   You may now begin using the Graphics Development Toolkit.

Checking Distribution Files

Following is a list of the files on the Graphics Development Toolkit diskette. These files will be transferred by the installation procedure to the directories indicated.

Graphics Development Toolkit requires approximately 800 blocks on the **/usr** minidisk. See *IBM RT PC Installing and Customizing the AIX Operating System* for additional information concerning minidisk size requirements and planning.

The language libraries reside in the directory **/usr/lpp/vdi/lib.** They are:

```
basvdi.a
cvdi.a
f77vdi.a
pasvdi.a
```

The following are include files for Pascal and BASIC programs. They reside in the **/usr/include** directory.

```
pasvdi.int
extrnvdi.bas
```

The demonstration program and its source file reside in **/usr/lpp/vdi/bin.** They are:

```
vdidemo.c
vdidemo
```

Additionally, the device driver files are loaded into the directory **/usr/lpp/vdi/drivers** by the Multi-User programs install. They are:

```
vdi3812
vdi4201
vdi5152
vdi5182
vdi6180
vdi7371
vdi7372
vdi7375
vdiacg
vdiamg
vdiega
vdiemg
vdigst
vdimeta
```

Setting Environmental Parameters

To use the Graphics Development Toolkit, you must export certain environmental parameters to the AIX Operating System. This can be done from the operating system, in either the "sh" or "csh" shell. Refer to the *IBM RT PC AIX Operating System Commands Reference* for a description of the "sh" and "csh" commands. The environmental commands can also be included in ".profile" (sh) or ".login" (csh) files which are executed each time a user logs in.

Your program diskette should already be in diskette drive /dev/rfd0. **Type Return** is the same as **Press Enter.** As installation continues, various files are listed on the screen as they are copied to the fixed disk.

8. When installation is complete, remove the program diskette from the diskette drive and replace it in its protective envelope in the binder.

   ```
   The installation process has completed.
   ```

9. Log off as super-user or as a member of the system group.

   You may now begin using the Graphics Development Toolkit.

Checking Distribution Files

Following is a list of the files on the Graphics Development Toolkit diskette. These files will be transferred by the installation procedure to the directories indicated.

Graphics Development Toolkit requires approximately 800 blocks on the **/usr** minidisk. See *IBM RT PC Installing and Customizing the AIX Operating System* for additional information concerning minidisk size requirements and planning.

The language libraries reside in the directory **/usr/lpp/vdi/lib.** They are:

```
basvdi.a
cvdi.a
f77vdi.a
pasvdi.a
```

The following are include files for Pascal and BASIC programs. They reside in the **/usr/include** directory.

```
pasvdi.int
extrnvdi.bas
```

The demonstration program and its source file reside in **/usr/lpp/vdi/bin.** They are:

```
vdidemo.c
vdidemo
```

Additionally, the device driver files are loaded into the directory **/usr/lpp/vdi/drivers** by the Multi-User programs install. They are:

```
vdi3812
vdi4201
vdi5152
vdi5182
vdi6180
vdi7371
vdi7372
vdi7375
vdiacg
vdiamg
vdiega
vdiemg
vdigst
vdimeta
```

Setting Environmental Parameters

To use the Graphics Development Toolkit, you must export certain environmental parameters to the AIX Operating System. This can be done from the operating system, in either the "sh" or "csh" shell. Refer to the *IBM RT PC AIX Operating System Commands Reference* for a description of the "sh" and "csh" commands. The environmental commands can also be included in ".profile" (sh) or ".login" (csh) files which are executed each time a user logs in.

1. Set the VDIPATH parameter. This parameter provides a path to the directory where the IBM RT PC Graphics Development Toolkit device driver files reside. The installation procedure for the IBM RT PC Multi-User programs places the device driver files in **/usr/lpp/vdi/drivers.** VDIPATH is set as follows:

   ```
   VDIPATH=/usr/lpp/vdi/drivers                          (sh shell)
   export VDIPATH
   ```

   (or)

   ```
   setenv VDIPATH /usr/lpp/vdi/drivers                   (csh shell)
   ```

2. Any Logical Device Names (device driver logical names) which will be referenced in the Open Workstation command must be assigned to the appropriate device driver file names. These logical names are user-selectable; the Graphics Development Toolkit has no pre-assigned logical names. The device driver files must be located in the directory specified by VDIPATH.

   The logical device name is assigned as follows:

   ```
   PRINTERA=vdi5182                                      (sh shell)
   export PRINTERA
   ```

   (or)

   ```
   setenv PRINTERA vdi5182                               (csh shell)
   ```

3. The device driver file names referenced in step 2 must be assigned to the system's physical devices.

   For example:

   ```
   vdi5182=/dev/tty3                                     (sh shell)
   export vdi5182
   ```

   (or)

   ```
   setenv vdi5182 /dev/tty3                              (csh shell)
   ```

Note: The **devices** program must be executed from the shell to establish tty ports for your graphics output devices. This program is native to the IBM AIX Operating System, but you must be a super-user to access it. The program prompts you for the required information. Refer to the *IBM RT PC Installing and Customizing the AIX Operating System* for information about how to run the **devices** program.

4. If you select a printer, you may also want to pipe the output through the system's spooler. To do this, enter the following in place of step 3.

   ```
   vdi5182='|print -plot lpn'
   export vdi5182
   ```
   (sh shell)

   (or)

   ```
   setenv vdi5182 '|print -plot lpn'
   ```
   (csh shell)

   Where lpn is the printer device name.

5. If you intend to send output to a plotter from a console display driver that is open, you must set a special environmental parameter called MESSAGEPORT. The MESSAGEPORT parameter may also be used to select a display device for the metafile "message" output.

   The default for this parameter is **/dev/tty**. If you do not set this parameter to **/dev/hft**, prompts from the plotter (such as those to change paper and pens) will not appear on the console. The commands to set MESSAGEPORT are:

   ```
   MESSAGEPORT=/dev/hft
   export MESSAGEPORT
   ```
   (sh shell)

   (or)

   ```
   setenv MESSAGEPORT /dev/hft
   ```
   (csh shell)

6. If you send output to a metafile (device driver file vdimeta), you may want to assign the output metafile file name to be used instead of the default, METAFILE.DAT.

   The commands to do this are:

   ```
   METAOUTPUT=filename                                    (sh shell)
   export METAOUTPUT
   ```

   (or)

   ```
   setenv METAOUTPUT filename                             (csh shell)
   ```

Compiling and Running the Demo Program

A demonstration program has been included with the IBM RT PC Graphics Development Toolkit. To run this program:

1. Ensure that the VDIPATH parameter is set correctly, as noted in step 1 of the procedure to set environmental parameters.

2. Select a graphics output device. The demonstration program uses logical device name DISPLAY. Ensure that the proper environmental parameters for this device have been set, as noted in steps 2 and 3 of the procedure to set environmental parameters.

3. Refer to Appendix C for a description of the device driver for the device you have selected. Set any specific environmental parameters required for this device.

4. To move to the appropriate directory, type:

   ```
   cd /usr/lpp/vdi/bin
   ```

   and press the **Enter** key.

5. To compile the demonstration program, type:

   ```
   cc -O vdidemo.c /usr/lpp/vdi/lib/cvdi.a -o vdidemo
   ```

   and press the **Enter** key.

6. To execute the demonstration program, type:

   ```
   vdidemo
   ```

   and press the **Enter** key.

Appendix B. Example Programs

The Graphics Development Toolkit distribution diskette contains both the source and executable files for a demonstration program written in the C language. This program is intended to demonstrate some of the capabilities of the Toolkit.

The executable demonstration program is:

/usr/lpp/vdi/bin/vdidemo

The demonstration's source file is:

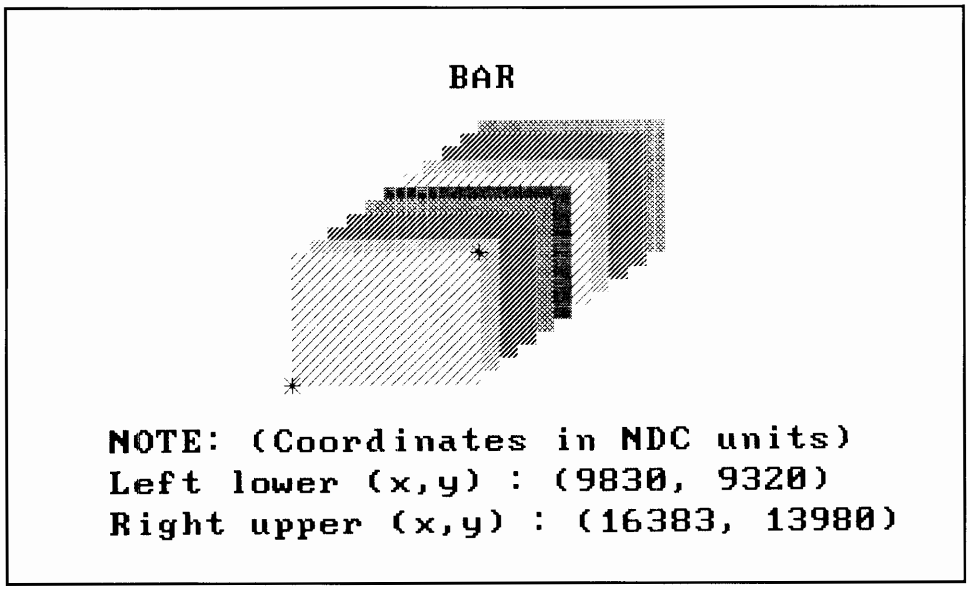
/usr/lpp/vdi/bin/vdidemo.c

Use one of the IBM RT PC AIX Operating System editing products to view the source file for the example program. To compile and run the example program, refer to the procedure at the end of Appendix A, "Installing the Graphics Development Toolkit."

This appendix contains some example programs that are not provided with the distribution diskette. The text of these examples must be entered with one of the editing products. The example programs use logical device name DISPLAY.

Throughout this appendix, each section begins with one or more figures that illustrate the output of some example program. Then, the code for that example program is listed.

Example 1 – Bar Function



```
NOTE: (Coordinates in NDC units)
Left lower (x,y) : (9830, 9320)
Right upper (x,y) : (16383, 13980)
```

Figure B-1. Bar Function, Part One

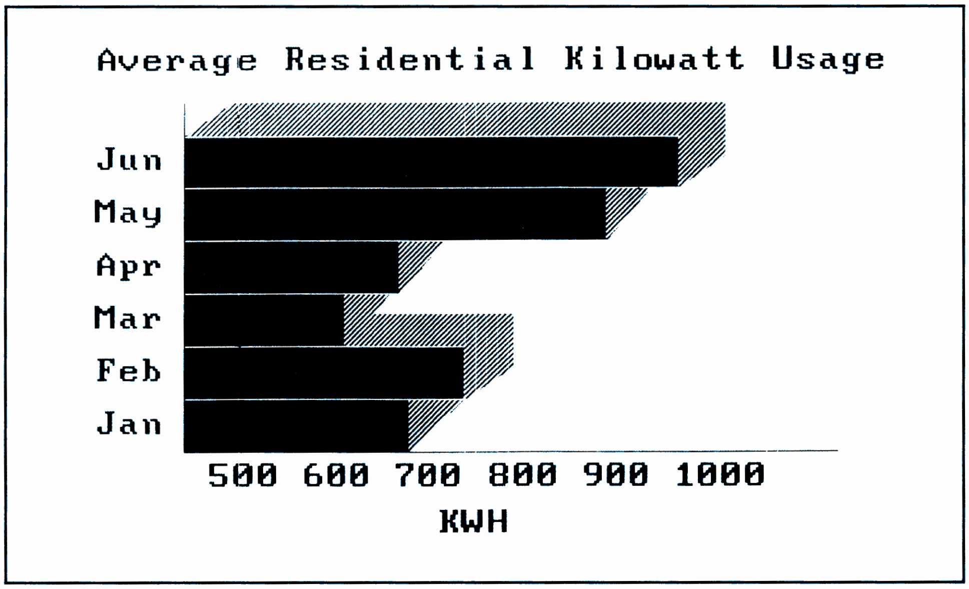


Figure B-2. Bar Function, Part Two

```
#include <stdio.h>


main()
/***********************************************/
/*                                             */
/*                                             */
/*   Example Program - Use of Bar Function     */
/*                                             */
/*                                             */
/***********************************************/
{
#define BLACK 0
#define WHITE 1

#define SOLID 1
#define HATCH 2

#define NO_ECHO 0
#define ECHO 1
```

```
#define NARROW_X 4
#define NARROW_DIAG 1

#define LEFT 0
#define CENTER 1
#define BOTTOM 0

extern short v_opnwk(), v_clrwk(), vst_height();
extern short vsf_interior(), vsf_style(),vsf_color();
extern short v_bar(), vsm_height(), v_pmarker();
extern short v_gtext(),vst_color(), vrq_string();
extern short vsf_color(), v_fillarea(), vsl_color();
extern short v_pline(), vrq_string(), v_enter_cur();
extern short vst_alignment(), v_clswk(), vq_error();


short savary[66], dev_handle;
static short savin[]={ 1,
                       1,
                       1,
                       3,
                       1,
                       1,
                       1,
                       0,
                       0,
                       1,
                       1,
                      'D''I','S','P','L','A','Y',' '};
static char *months[] = { "Jan",
                          "Feb",
                          "Mar",
                          "Apr",
                          "May",
                          "Jun" };

short next_line, xwid, cwid, chgt, tx, ty, count;
short pc15y, pc22y, pc27y, pc30y;
short pc35y, pc40y, pc43y, pc48y;
short pc51y, pc59y, pc67y;
short pc72y, pc75y, pc80y, pc85y, pc95y;

short pc10x, pc20x, pc25x, pc37x, pc40x;
short pc42x, pc43x, pc44x, pc47x, pc48x;
short pc50x, pc55x, pc65x, pc70x, pc73x;
short pc78x, pc90x;

short echo_xy[2], xy[12];
short horz_out, vert_out, tmp, xaxis, yaxis, i;
char tstr[6];
char label[81];

   echo_xy[0] = 0;
   echo_xy[1] = 0;

   /* Open the Workstation */
   if(v_opnwk( savin, &dev_handle, savary ) < 0 ){
      error_handler();
      exit(0);
   }

   if ( savary[14] != 0 ){  /* GDP's Available */
      for ( i = savary[14] ; i >= 0 ; i-- ){
         if ( savary[i + 15] == 1 ) break;
      }

      /* If bar gdp not available, */
      /* close the workstation.    */
      if ( i >= 0 ){
         /*  savary[51] = max. NDC space - x axis */
         /*  savary[52] = max. NDC space - y axis */
         xaxis = savary[51];
         yaxis = savary[52];
```

```
/* set up tags as %'s of max. x and y axes */
pc15y = (yaxis / 100) * 15 ;
pc22y = (yaxis / 100) * 22 ;
pc27y = (yaxis / 100) * 27 ;
pc30y = (yaxis / 100) * 30 ;
pc35y = (yaxis / 100) * 35 ;
pc40y = (yaxis / 100) * 40 ;
pc43y = (yaxis / 100) * 43 ;
pc48y = (yaxis / 100) * 48 ;
pc51y = (yaxis / 100) * 51 ;
pc59y = (yaxis / 100) * 59 ;
pc67y = (yaxis / 100) * 67 ;
pc72y = (yaxis / 100) * 72 ;
pc75y = (yaxis / 100) * 75 ;
pc80y = (yaxis / 100) * 80 ;
pc85y = (yaxis / 100) * 85 ;
pc95y = (yaxis / 100) * 95 ;

pc10x = (xaxis / 100) * 10 ;
pc20x = (xaxis / 100) * 20 ;
pc25x = (xaxis / 100) * 25 ;
pc37x = (xaxis / 100) * 37 ;
pc40x = (xaxis / 100) * 40 ;
pc42x = (xaxis / 100) * 42 ;
pc43x = (xaxis / 100) * 43 ;
pc44x = (xaxis / 100) * 44 ;
pc47x = (xaxis / 100) * 47 ;
pc48x = (xaxis / 100) * 48 ;
pc50x = (xaxis / 100) * 50 ;
pc55x = (xaxis / 100) * 55 ;
pc65x = (xaxis / 100) * 65 ;
pc70x = (xaxis / 100) * 70 ;
pc73x = (xaxis / 100) * 73 ;
pc78x = (xaxis / 100) * 78 ;
pc90x = (xaxis / 100) * 90 ;
```

```
/* first display = bar primitive */
if (v_clrwk(dev_handle) < 0 )
   error_handler();

/* set requested text height = 1600 NDC units */
if(vst_height(dev_handle,
              1600,
              &xwid,
              &cwid,
              &chgt) <0)
   error_handler() ;

/* set requested fill interior = hatch     */
if(vsf_interior(dev_handle, HATCH ) < 0 )
   error_handler();

/* set fill color */
if(vsf_color(dev_handle, WHITE) < 0 )
   error_handler();

for ( i = 1 ; i <= 11 ; i++ ){

   /* set hatch index */
   if(vsf_style(dev_handle, (i + 4) % 6) < 0)
      error_handler();

   xy[0] =
   (short)((0.5 - ((float)(i - 1) * 0.02)) *
   (float)xaxis);
   xy[1] =
   (short)((0.6 - ((float)(i - 1) * 0.02)) *
   (float)yaxis);
   xy[2] =
   (short)((0.7 - ((float)(i - 1) * 0.02)) *
   (float)xaxis);
   xy[3] =
   (short)((0.8 - ((float)(i - 1) * 0.02)) *
   (float)yaxis);

   /* output bar */
   if(v_bar(dev_handle, xy) < 0)
      error_handler();
}
```

```
/* output 2 polymarkers (stars) to     */
/* mark bar corners                    */
/* set marker height to 800 NDC units */
if(vsm_height(dev_handle, 800) < 0 )
   error_handler();
if(v_pmarker(dev_handle, 2, xy) < 0 )
   error_handler();

/* label drawing with text */
if (vst_alignment(dev_handle,
                  CENTER,
                  BOTTOM,
                  &horz_out,
                  &vert_out) < 0 )
   error_handler();

if (v_gtext(dev_handle,
            pc50x,
            pc85y ,
            "BAR") < 0 )
   error_handler();

if (vst_alignment(dev_handle,
                  LEFT,
                  BOTTOM,
                  &horz_out,
                  &vert_out) < 0 )
   error_handler();

next_line = pc30y;
if (v_gtext(
       dev_handle,
       pc10x,
       next_line,
       "NOTE: (Coordinates in NDC units)") < 0 )
   error_handler();
```

```
/* build label strings and output it */
sprintf( label,
         "Left lower (x,y) : (%d, %d)",
         xy[0],
         xy[1]);

next_line -=chgt;
if (v_gtext(dev_handle,
            pc10x,
            next_line,
            label ) < 0 )
   error_handler();

sprintf( label,
         "Right upper (x,y) : (%d, %d)",
         xy[2],
         xy[3]);

next_line -=chgt;
if (v_gtext(dev_handle,
            pc10x,
            next_line,
            label ) < 0 )
   error_handler();

/* wait for keystroke to continue */
if(vrq_string(dev_handle,
              1,
              NO_ECHO,
              echo_xy,
              tstr) < 0 )

   error_handler();

/*   2nd display - bar graph */
if (v_clrwk(dev_handle) < 0 )
   error_handler();
```

```
/* label x axis of graph with KWH values */
for ( i = 0 ; i <= 5 ; i++ ){

  tx =
  (short)(( 0.22 + ((float)i * 0.1)) *
  (float)xaxis);

  tmp = (i + 5) * 100;
  sprintf(label,"%d", tmp);
  if(v_gtext(dev_handle,
             tx,
             pc22y,
             label) < 0 )
     error_handler();
}

/* label the y axis of graph with months */
for (i = 0 ; i < 6 ; i++ ){
   ty =
   (short)((0.3 + ((float)i * 0.08)) *
   (float)yaxis);

   if(v_gtext(dev_handle,
              pc10x,
              ty,
              months[i]) < 0 )
      error_handler();
}

/* set fill interior style to solid */
/* and color to magenta            */
if(vsf_interior(dev_handle, SOLID) < 0 )
   error_handler();

/* output the bars to graph */
xy[0] = pc20x;  /* doesnt change */

xy[1] = pc67y;
xy[2] = pc73x;
xy[3] = pc75y;
if(v_bar(dev_handle, xy) < 0)
   error_handler();
```

```
xy[1] = pc59y;
xy[2] = pc65x;
xy[3] = pc67y;
if(v_bar(dev_handle, xy) < 0)
   error_handler();

xy[1] = pc51y;
xy[2] = pc43x;
xy[3] = pc59y;
if(v_bar(dev_handle, xy) < 0)
   error_handler();

xy[1] = pc43y;
xy[2] = pc37x;
xy[3] = pc51y;
if(v_bar(dev_handle, xy) < 0)
   error_handler();

xy[1] = pc35y;
xy[2] = pc50x;
xy[3] = pc43y;
if(v_bar(dev_handle, xy) < 0)
   error_handler();

xy[1] = pc27y;
xy[2] = pc44x;
xy[3] = pc35y;
if(v_bar(dev_handle, xy) < 0)
   error_handler();

/* change fill = hatch , narrow diagonal  */
if(vsf_interior(dev_handle, HATCH) < 0 )
   error_handler();
if (vsf_style(dev_handle, NARROW_DIAG) < 0 )
   error_handler();
```

```
/* output filled areas  */
/* area has 4 corners */
count = 4;
xy[0] = pc20x;
xy[1] = pc75y;
xy[2] = pc25x;
xy[3] = pc80y;
xy[4] = pc78x;
xy[5] = pc80y;
xy[6] = pc73x;
xy[7] = pc75y;
if(v_fillarea(dev_handle, count, xy) < 0 )
   error_handler();

xy[0] = pc37x;
xy[1] = pc43y;
xy[2] = pc40x;
xy[3] = pc48y;
xy[4] = pc55x;
xy[5] = pc48y;
xy[6] = pc50x;
xy[7] = pc43y;
if(v_fillarea(dev_handle, count, xy) < 0 )
   error_handler();

xy[0] = pc73x;
xy[1] = pc67y;
xy[2] = pc73x;
xy[3] = pc75y;
xy[4] = pc78x;
xy[5] = pc80y;
xy[6] = pc78x;
xy[7] = pc72y;
if(v_fillarea(dev_handle, count, xy) < 0 )
   error_handler();
```

```
/* next set of areas have 3 corners */
count = 3;

xy[0] = pc65x;
xy[1] = pc67y;
xy[2] = pc65x;
xy[3] = pc59y;
xy[4] = pc70x;
xy[5] = pc67y;
if(v_fillarea(dev_handle, count, xy) < 0 )
   error_handler();

xy[0] = pc43x;
xy[1] = pc59y;
xy[2] = pc43x;
xy[3] = pc51y;
xy[4] = pc48x;
xy[5] = pc59y;
if(v_fillarea(dev_handle, count, xy) < 0 )
   error_handler();

xy[0] = pc37x;
xy[1] = pc51y;
xy[2] = pc37x;
xy[3] = pc43y;
xy[4] = pc42x;
xy[5] = pc51y;
if(v_fillarea(dev_handle, count, xy) < 0 )
   error_handler();
```

```
/* count goes back to 4 */
count = 4;

xy[0] = pc50x;
xy[1] = pc43y;
xy[2] = pc55x;
xy[3] = pc48y;
xy[4] = pc55x;
xy[5] = pc40y;
xy[6] = pc50x;
xy[7] = pc35y;
if(v_fillarea(dev_handle, count, xy) < 0 )
   error_handler();

/* three sided once more */
count = 3;

xy[0] = pc44x;
xy[1] = pc35y;
xy[2] = pc44x;
xy[3] = pc27y;
xy[4] = pc50x;
xy[5] = pc35y;
if(v_fillarea(dev_handle, count, xy) < 0 )
   error_handler();

/* set the color to background */
if (vsl_color(dev_handle, BLACK)< 0 )
   error_handler();
```

```
/* output polylines to delineate bars */
count = 2;

xy[0] = pc20x; /* remains same for awhile */

xy[1] = pc75y;
xy[2] = pc73x;
xy[3] = pc75y;
if(v_pline(dev_handle, count, xy) < 0 )
   error_handler();

xy[1] = pc67y;
xy[2] = pc73x;
xy[3] = pc67y;
if(v_pline(dev_handle, count, xy) < 0 )
   error_handler();

xy[1] = pc59y;
xy[2] = pc65x;
xy[3] = pc59y;
if(v_pline(dev_handle, count, xy) < 0 )
   error_handler();

xy[1] = pc51y;
xy[2] = pc43x;
xy[3] = pc51y;
if(v_pline(dev_handle, count, xy) < 0 )
   error_handler();

xy[1] = pc43y;
xy[2] = pc50x;
xy[3] = pc43y;
if(v_pline(dev_handle, count, xy) < 0 )
   error_handler();
```

```
xy[1] = pc35y;
xy[2] = pc50x;
xy[3] = pc35y;
if(v_pline(dev_handle, count, xy) < 0 )
   error_handler();

xy[1] = pc27y;
xy[2] = pc44x;
xy[3] = pc27y;
if(v_pline(dev_handle, count, xy) < 0 )
   error_handler();

if (v_gtext(dev_handle,
       pc10x,
       pc85y,
       "Average Residential Kilowatt Usage")
       < 0 )
    error_handler();

 if (v_gtext(dev_handle,
             pc47x,
             pc15y,
             "KWH") < 0 )
    error_handler();

/* reset line color */
if (vsl_color(dev_handle, WHITE)< 0 )
   error_handler();

/* output polylines for graph axes */
count = 3;

xy[0] = pc20x;
xy[1] = pc80y;
xy[2] = pc20x;
xy[3] = pc27y;
xy[4] = pc90x;
xy[5] = pc27y;
if(v_pline(dev_handle, count, xy) < 0 )
   error_handler();
```

```
         /* wait for keystroke to continue */
         if(vrq_string(dev_handle,
                       1,
                       NO_ECHO,
                       echo_xy,
                       tstr) < 0 )
            error_handler();

      }
      else
         printf(
            "Bar GDP not available on this device\n\r"
         );

   }
   else{
     printf(" No GDP's available on this device\n\r");
   }

   /* close the workstation */
   if (v_enter_cur(dev_handle) < 0 )
      error_handler();
   if (v_clswk(dev_handle) < 0 )
      error_handler();

}


/****************************************/
/*                                      */
          error_handler()
/*                                      */
/****************************************/
{
   extern short vq_error();

   printf(" GDT error, number = %d\n\r", vq_error());
}
```

Example 2 – Graphic Text Function

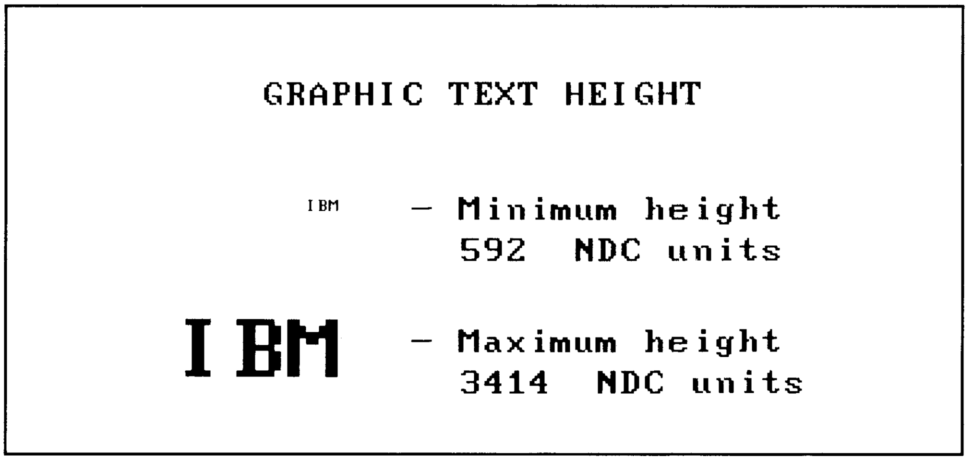
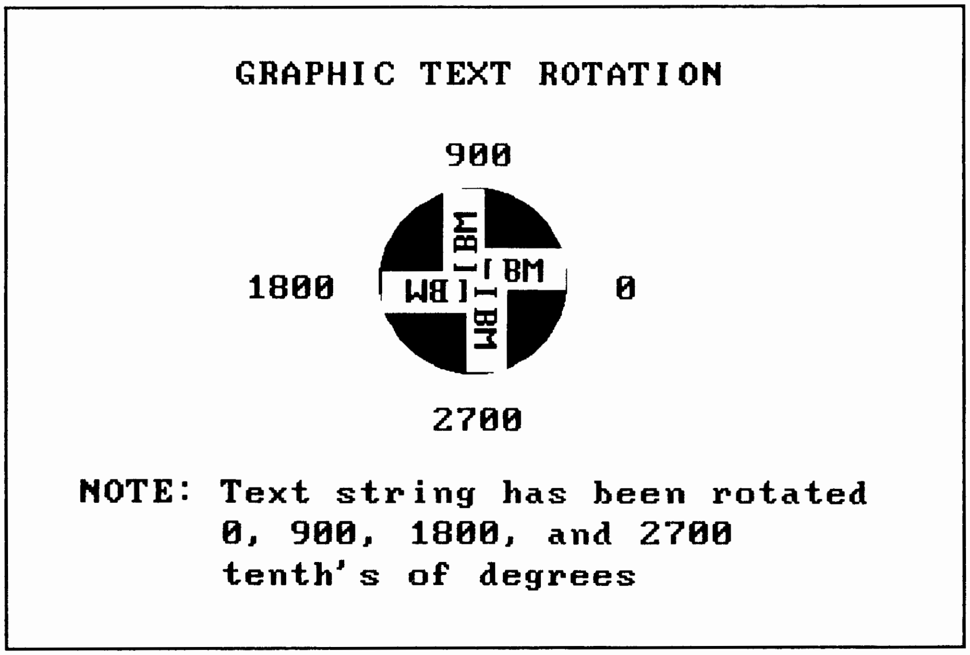


Figure B-3. Graphic Text Function, Height


GRAPHIC TEXT ROTATION

900

1800

0

2700

NOTE: Text string has been rotated
      0, 900, 1800, and 2700
      tenth's of degrees

Figure B-4. Graphic Text Function, Rotation

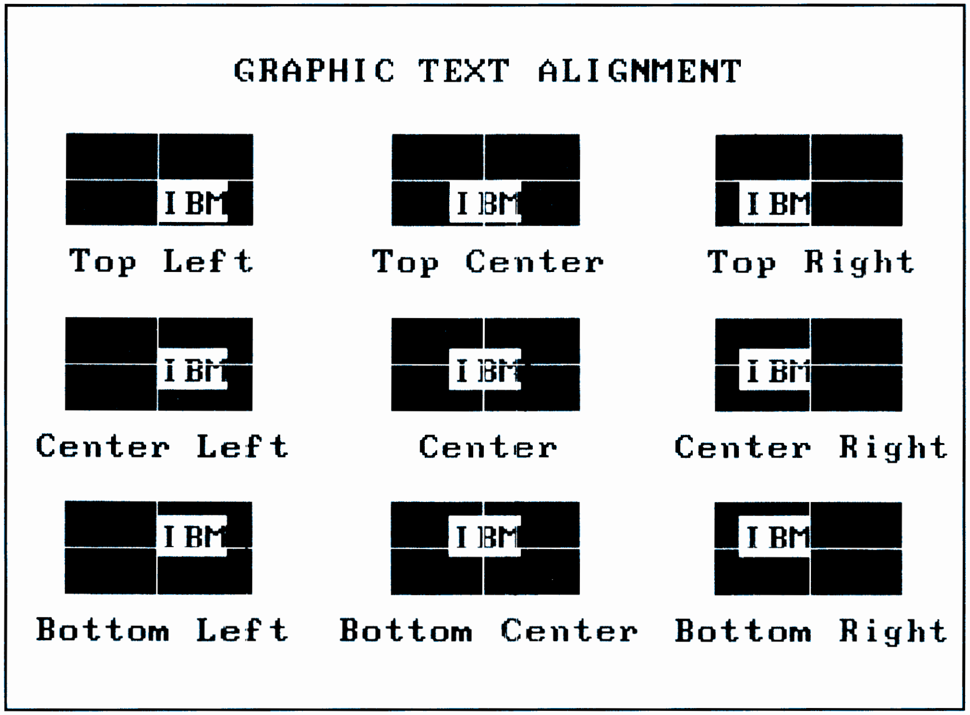


Figure B-5. Graphic Text Function, Alignment

```
#include <stdio.h>

#define NO_ECHO 0
#define ECHO 1

#define BLACK 0
#define WHITE 1
#define CYAN  6
#define MAGENTA 7

#define SOLID 1

#define CENTER 1
#define LEFT 0
#define RIGHT 2
```

```
#define BOTTOM 0
#define TOP 2


short dev_handle;

main()
/************************************************/
/*                                              */
/*                                              */
/*   Example Program - Use of Graphic Text      */
/*                                              */
/*                                              */
/************************************************/
{
extern short v_opnwk(), v_clrwk(), vst_height();
extern short v_gtext(), vst_color();
extern short vsf_color(), vsf_interior();
extern short v_circle(), vst_rotation();
extern short vsl_color(), v_bar(), v_pline();
extern short v_enter_cur(), v_clswk();
extern short vst_alignment();


short savary[66];
static short savin[] = { 1,
                         1,
                         1,
                         3,
                         1,
                         1,
                         1,
                         0,
                         0,
                         1,
                         1,
                       'D','I','S','P','L','A','Y',' '};

static char *halign[] = { "Left", "Center", "Right"};
static char *valign[] = { "Bottom", "Center", "Top"};
short minhgt, maxhgt, xwid, cwid, chgt, tx, ty;
short savxy2, savxy0, savxy1, savxy3, hreq, vreq, hsel;
short hgt, centerx, centery, vsel, radius2, radius;

short pc05y, pc07y, pc20y, pc25y, pc28y, pc50y;
short pc60y, pc70y, pc75y, pc85y, pc90y, pc95y;
short pc10x, pc15x, pc35x, pc40x, pc50x;

short xy[12];
short xaxis, yaxis, i, j;
char *ptr;
char label[81];


   /* open the workstation */
   if(v_opnwk( savin, &dev_handle, savary ) >= 0 ){

      /*  savary[51] = max. NDC space - x axis */
      /*  savary[52] = max. NDC space - y axis */
      xaxis = savary[51];
      yaxis = savary[52];

      minhgt = savary[60];
      maxhgt = savary[61];

      /* set up tags as %'s of max. x and y axes */
      pc05y = (yaxis / 100) * 5;
      pc07y = (yaxis / 100) * 7;
      pc20y = (yaxis / 100) * 20;
      pc25y = (yaxis / 100) * 25;
      pc28y = (yaxis / 100) * 28;
      pc50y = (yaxis / 100) * 50;
      pc60y = (yaxis / 100) * 60;
      pc70y = (yaxis / 100) * 70;
      pc75y = (yaxis / 100) * 75;
      pc85y = (yaxis / 100) * 85;
      pc90y = (yaxis / 100) * 90;
      pc95y = (yaxis / 100) * 95;

      pc10x = (xaxis / 100) * 10;
      pc15x = (xaxis / 100) * 15;
      pc35x = (xaxis / 100) * 35;
      pc40x = (xaxis / 100) * 40;
      pc50x = (xaxis / 100) * 50;
```

```
/***************************/
/*                         */
/* demonstrate text height */
/*                         */
/***************************/

 if (v_clrwk(dev_handle) < 0 )
    error_handler();

 /* set the text alignment */
 if(vst_alignment(dev_handle,
                  RIGHT,
                  CENTER,
                  &hsel,
                  &vsel) < 0 )
    error_handler();

 /* set maximum height */
 if(vst_height(dev_handle,
               maxhgt,
               &xwid,
               &cwid,
               &chgt) < 0 )
    error_handler();

 if(v_gtext(dev_handle, pc35x, pc50y, "IBM") < 0)
    error_handler();

 /* set minimum height */
 if(vst_height(dev_handle,
               minhgt,
               &xwid,
               &cwid,
               &chgt) < 0 )
    error_handler();

 if(v_gtext(dev_handle, pc35x, pc70y, "IBM") < 0)
    error_handler();
```

```
/* set text height to 1600 NDC units */
if(vst_height(dev_handle,
              1600,
              &xwid,
              &cwid,
              &chgt) < 0 )
   error_handler();

/* set the text alignment */
if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &hsel,
                 &vsel) < 0 )
   error_handler();

/* output label */
if(v_gtext(dev_handle,
           pc50x,
           pc90y,
           "GRAPHIC TEXT HEIGHT") < 0)
   error_handler();

/* set the text alignment */
if(vst_alignment(dev_handle,
                 LEFT,
                 CENTER,
                 &hsel,
                 &vsel) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           pc40x,
           pc70y,
           " - Minimum height") < 0)
   error_handler();

sprintf( label,"   %d  NDC units", minhgt );
if(v_gtext(dev_handle,
           pc40x,
           pc70y - chgt,
           label) < 0)
   error_handler();
```

```
 if(v_gtext(dev_handle,
            pc40x,
            pc50y,
            " - Maximum height") < 0)
    error_handler();

 sprintf( label,"   %d  NDC units", maxhgt );
 if(v_gtext(dev_handle,
            pc40x,
            pc50y - chgt,
            label) < 0)
    error_handler();

 wait_kybrd();

/******************************/
/*                            */
/* demonstrate text rotation  */
/*                            */
/******************************/

 if (v_clrwk(dev_handle) < 0 )
    error_handler();

 /* if graphic text rotation not available */
 /* on device then skip                    */
 if ( savary[36] ){

    /* set text color */
    if(vst_color(dev_handle, WHITE) < 0 )
       error_handler();

    /* set text height to 1595 NDC units */
    if(vst_height(dev_handle,
                  1595,
                  &xwid,
                  &cwid,
                  &chgt) < 0 )
       error_handler();

    /* filled circle radius is 4 text cells wide */
    radius = (4 * cwid);
```

```
/* set the text alignment */
if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &hsel,
                 &vsel) < 0 )
   error_handler();

/* set text height to 1600 NDC units */
if(vst_height(dev_handle,
              1600,
              &xwid,
              &cwid,
              &chgt) < 0 )
   error_handler();

/* position labels two cell widths  */
/* beyond circle                    */
radius2 = radius + (2 * cwid);

if(v_gtext(dev_handle,
           pc50x,
           pc95y,
           "GRAPHIC TEXT ROTATION") < 0)
   error_handler();

/* set the text alignment */
if(vst_alignment(dev_handle,
                 CENTER,
                 BOTTOM,
                 &hsel,
                 &vsel) < 0 )
   error_handler();

hgt = pc60y - (radius2 + (2 * chgt));

if(v_gtext(
    dev_handle,
    pc50x,
    hgt,
    "NOTE: Text string has been rotated") < 0)
   error_handler();

hgt -= chgt;
```

```
if(v_gtext(
   dev_handle,
   pc50x,
   hgt,
   "      0, 900, 1800, and 2700      ") < 0)
   error_handler();

hgt -= chgt;
if(v_gtext(dev_handle,
   pc50x,
   hgt,
   "      tenth's of degrees          ") < 0)
   error_handler();

/* set fill interior */
if(vsf_interior(dev_handle, SOLID) < 0)
   error_handler();

/* output background circle */
centerx = pc50x;
centery = pc60y;
if(v_circle(dev_handle,
            centerx,
            centery,
            radius ) < 0 )
   error_handler();

/* label 0, 900, 1800, 2700 tenth's of */
/* degree positions on circle          */
/* set the text alignment              */
if(vst_alignment(dev_handle,
                 LEFT,
                 CENTER,
                 &hsel,
                 &vsel) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           centerx + radius2 ,
           centery,"0") < 0)
   error_handler();
```

```
/* set the text alignment */
if(vst_alignment(dev_handle,
                 CENTER,
                 BOTTOM,
                 &hsel,
                 &vsel) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           centerx,
           centery + radius +cwid,
           "900") < 0)
   error_handler();

/* set the text alignment */
if(vst_alignment(dev_handle,
                 RIGHT,
                 CENTER,
                 &hsel,
                 &vsel) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           centerx - radius2,
           centery,
           "1800") < 0)
   error_handler();

/* set the text alignment */
if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &hsel,
                 &vsel) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           centerx,
           centery - (radius + cwid),
           "2700") < 0)
   error_handler();
```

```
      /* set the text alignment */
      if(vst_alignment(dev_handle,
                       LEFT,
                       BOTTOM,
                       &hsel,
                       &vsel) < 0 )
         error_handler();

      /* set text height to 1595 NDC units */
      if(vst_height(dev_handle,
                    1595,
                    &xwid,
                    &cwid,
                    &chgt) < 0 )
         error_handler();

      /* rotate "IBM" 0, 900, 1800, 2700  */
      /* tenth's of degrees                */
      for ( i = 0 ; i <= 2700 ; i+=900 ){
         if(vst_rotation(dev_handle,i) < 0 )
            error_handler();
         if(v_gtext(dev_handle,
                    centerx,
                    centery,
                    "IBM ") < 0)
            error_handler();
      }
      /* reset rotation back to 0 */
      if(vst_rotation(dev_handle,0) < 0 )
         error_handler();

   }
   else{
      printf("Graphic Text cannot be rotated");
      printf(" on this device\n\r");
      printf("Press any key\n\r");
   }

   wait_kybrd();
```

```
/******************************/
/*                            */
/* demonstrate text alignment */
/*                            */
/******************************/
 if (v_clrwk(dev_handle) < 0 )
    error_handler();

 /* set text height to 1600 NDC units */
 if(vst_height(dev_handle,
               1600,
               &xwid,
               &cwid, &chgt) < 0 )
    error_handler();

 /* set the text alignment */
 if(vst_alignment(dev_handle,
                  CENTER,
                  TOP,
                  &hsel,
                  &vsel) < 0 )
    error_handler();

 if(v_gtext(dev_handle,
            pc50x,
            pc95y,
            "GRAPHIC TEXT ALIGNMENT") < 0)
    error_handler();

 /* set fill interior */
 if(vsf_interior(dev_handle, SOLID) < 0)
    error_handler();

 /* set line color */
 if(vsl_color(dev_handle, BLACK) < 0 )
    error_handler();

 /* show nine cases of text alignment */
 for ( i = 0 ; i < 3 ; i++ ){
    ty = pc75y - (i * pc28y);
    savxy1 = ty - pc07y;
    savxy3 = ty + pc07y;
```

```
for ( j = 0 ; j < 3 ; j++ ){
   tx = (j * pc35x) + pc15x;
   xy[0] = tx - pc10x;
   xy[2] = tx + pc10x;
   savxy0 = xy[0] ;
   savxy2 = xy[2] ;
   xy[1] = savxy1 ;
   xy[3] = savxy3 ;

   /* output bars */
   if(v_bar(dev_handle, xy) < 0)
      error_handler();


   hreq = j;
   if(i == 2 ) vreq = 0;
   else if(i == 0 ) vreq = 2;
   else vreq = i;

   /* set the text alignment */
   if(vst_alignment(dev_handle,
                    hreq,
                    vreq,
                    &hsel,
                    &vsel) < 0 )
      error_handler();

   if(v_gtext(dev_handle, tx, ty, "IBM") < 0)
      error_handler();

   /* output crossed polylines */
   xy[0] = xy[2] = tx;
   if(v_pline(dev_handle, 2, xy) < 0 )
      error_handler();

   xy[0] = savxy0;
   xy[2] = savxy2;
   xy[1] = xy[3] = ty ;
   if(v_pline(dev_handle, 2, xy) < 0 )
      error_handler();
```

```
                /* label cases for alignment */
                if((hreq == 1) && (vreq == 1)){
                   ptr = "Center";
                }
                else{
                   ptr = label;
                   sprintf(label,
                           "%s %s",
                           valign[vreq],
                           halign[hreq]);
                }

                if(vst_alignment(dev_handle,
                                 1,
                                 0,
                                 &hsel,
                                 &vsel) < 0 )
                   error_handler();

                if(v_gtext(dev_handle,
                           tx,
                           savxy1-pc07y,
                           ptr) < 0)
                   error_handler();
             }
          }

          wait_kybrd();

          /* close the workstation */
          if (v_enter_cur(dev_handle) < 0 )
             error_handler();
          if (v_clswk(dev_handle) < 0 )
             error_handler();
       }
       else
         error_handler();

    }
```

```
/***************************************/
/*                                     */
             wait_kybrd()
/*                                     */
/***************************************/
{
  extern short vrq_string();
  short xy[2];
  char ch;

  if(vrq_string(dev_handle, 1, NO_ECHO, xy, &ch) < 0 )
     error_handler();
}


/***************************************/
/*                                     */
           error_handler()
/*                                     */
/***************************************/
{
   extern short vq_error();
   printf("GDT error, number = %d\n\r", vq_error());
}
```

Example 3 – Pie Slice Function

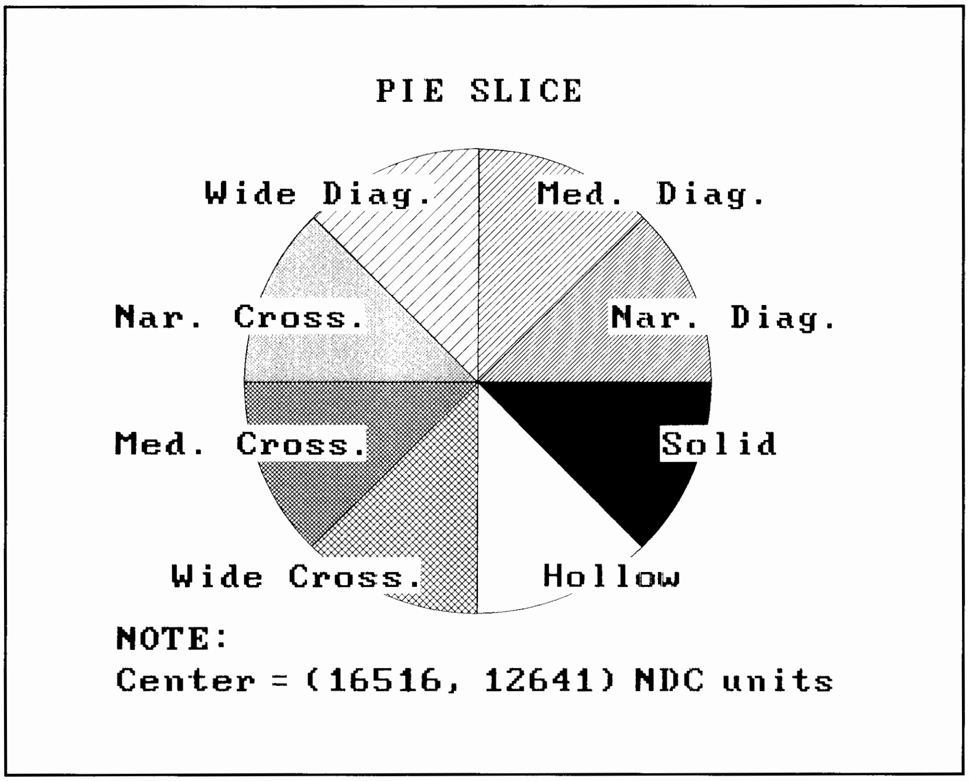


Figure B-6. Pie Slice, Part One

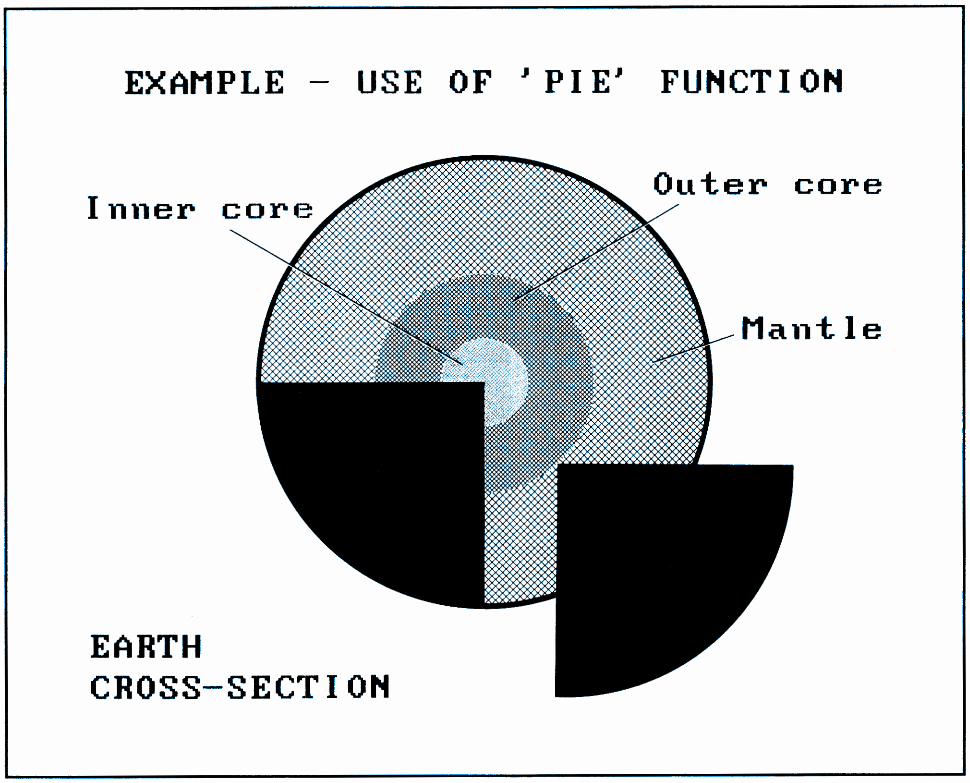


Figure B-7. Pie Slice, Part Two

```
#include <stdio.h>
#include <math.h>

#define NO_ECHO 0
#define ECHO 1

#define BLACK 0
#define WHITE 1

#define SOLID 1
#define HATCH 3

#define WIDE_X 6
#define MED_X 5
#define NARROW_X 4
```

```
#define LEFT 0
#define RIGHT 2
#define BOTTOM 0
#define TOP 2
#define CENTER 1

short dev_handle, xaxis, yaxis, max_index;
short align, thgt, strt_ang, end_ang;


short pc02y, pc05y, pc10y, pc15y, pc17y, pc40y;
short pc50y, pc54y, pc55y, pc65y, pc70y;
short pc80y, pc85y, pc90y, pc95y;
short pc02x, pc05x, pc10x, pc12x, pc12_5x;
short pc15x, pc17x, pc22_5x, pc24x, pc25x, pc40x;
short pc47_5x, pc49x, pc50x, pc55x, pc60x, pc65x;
short pc70x, pc72_5x, pc73x, pc75x, pc80x;


main()
/************************************************/
/*                                              */
/*                                              */
/*  Example Program - Use of Pie Slice Function */
/*                                              */
/*                                              */
/************************************************/
{
extern short v_opnwk(), v_clrwk(), vst_height();
extern short vsf_interior(), vsf_style();
extern short v_circle(),vsf_color(),v_bar();
extern short vsm_height(),v_pmarker(), v_gtext();
extern short vst_color(), v_pieslice(), vsf_color();
extern short v_fillarea(), vsl_color();
extern short v_pline(), vrq_string(), v_enter_cur();
extern short v_clswk(), vq_error();

short savary[66];
```

```
static short savin[]={ 1,
                       1,
                       1,
                       3,
                       1,
                       1,
                       1,
                       0,
                       0,
                       1,
                       1,
                       'D','I','S','P','L','A','Y',' '};

short xwid, cwid, chgt;
short centerx, centery, radius;

short xy[14];
short i;
char label[81];


   /* open the workstation */
   if(v_opnwk( savin, &dev_handle, savary ) >= 0 ){

      if ( savary[14] != 0 ){ /* gdp's available ? */

         /* see if pie slices supported */
         for ( i = savary[14] ; i >= 0 ; i-- ){
            if ( savary[i + 15] == 3 ) break;
         }
         /* if pie slices not supported  */
         /* go to close workstation      */
         if ( i >= 0 ){

            /*  savary[51] = max. NDC space - x axis */
            /*  savary[52] = max. NDC space - y axis */
            xaxis = savary[51];
            yaxis = savary[52];
            /* set up tags as %'s of  */
            /* max. x and y axes      */
            pc02y = (yaxis / 100 ) * 2;
            pc05y = (yaxis / 100 ) * 5;
            pc10y = (yaxis / 100 ) * 10;
            pc15y = (yaxis / 100 ) * 15;
            pc17y = (yaxis / 100 ) * 17;
            pc40y = (yaxis / 100 ) * 40;
```

```
pc50y = (yaxis / 100 ) * 50;
pc54y = (yaxis / 100 ) * 54;
pc55y = (yaxis / 100 ) * 55;
pc65y = (yaxis / 100 ) * 65;
pc70y = (yaxis / 100 ) * 70;
pc80y = (yaxis / 100 ) * 80;
pc85y = (yaxis / 100 ) * 85;
pc90y = (yaxis / 100 ) * 90;
pc95y = (yaxis / 100 ) * 95;

pc02x = (xaxis / 100 ) * 2;
pc05x = (xaxis / 100 ) * 5;
pc10x = (xaxis / 100 ) * 10;
pc12x = (xaxis / 100 ) * 12;
pc12_5x = (xaxis / 1000 ) * 125;
pc15x = (xaxis / 100 ) * 15;
pc17x = (xaxis / 100 ) * 17;
pc22_5x = (xaxis / 1000 ) * 225;
pc24x = (xaxis / 100 ) * 24;
pc25x = (xaxis / 100 ) * 25;
pc40x = (xaxis / 100 ) * 40;
pc47_5x = (xaxis / 1000 ) * 475;
pc49x = (xaxis / 100 ) * 49;
pc50x = (xaxis / 100 ) * 50;
pc55x = (xaxis / 100 ) * 55;
pc60x = (xaxis / 100 ) * 60;
pc65x = (xaxis / 100 ) * 65;
pc70x = (xaxis / 100 ) * 70;
pc72_5x = (xaxis / 1000 ) * 725;
pc73x = (xaxis / 100 ) * 73;
pc75x = (xaxis / 100 ) * 75;
pc80x = (xaxis / 100 ) * 80;

/* output 8 sets of pie slices with */
/* varying interior styles          */
if (v_clrwk(dev_handle) < 0 )
   error_handler();

if(vst_height(dev_handle,
              1600,
              &xwid,
              &cwid,
              &chgt) < 0)
   error_handler();
```

```
     /* set up device max. color index */
     max_index = savary[39] - 1;

     /* output pie slice */
     centerx = pc50x + cwid - pc02x;
     centery = pc50y + chgt - pc02y;
     pie_slice(centerx, centery);

    /* set alignment */
    if( vst_alignment(dev_handle,
                      CENTER,
                      TOP,
                      &align,
                      &align ) < 0 )
       error_handler();

     if(v_gtext(dev_handle,
                pc50x,
                yaxis,
                "PIE SLICE") < 0 )
        error_handler();

    /* set alignment */
    if( vst_alignment(dev_handle,
                      LEFT,
                      BOTTOM,
                      &align,
                      &align ) < 0 )
       error_handler();

     /* label drawings */
     thgt = 2 * chgt;
     if(v_gtext(dev_handle,
                     pc05x,
                     thgt,
                     "NOTE:") < 0 )
        error_handler();

     sprintf( label,
              "Center = (%d, %d) NDC units",
              centerx,
              centery );

     thgt -= chgt;
```

```
if(v_gtext(dev_handle,
                pc05x,
                thgt,
                label) < 0 )
   error_handler();

wait_kybrd();

/************************************/
/*                                  */
/*  use pie slice function to draw  */
/*  cross-section of the earth      */
/*                                  */
/************************************/
if (v_clrwk(dev_handle) < 0 )
   error_handler();

/* draw earth (outer circle )  */
/* set fill to solid  */
if(vsf_interior(dev_handle, SOLID) < 0)
   error_handler();

radius = pc25x;
centerx = pc50x;
centery = pc50y;

if(v_circle(dev_handle,
            centerx,
            centery,
            radius ) < 0 )
   error_handler();

/* set fill to hatch  */
if(vsf_interior(dev_handle, HATCH) < 0)
   error_handler();

/* draw mantle - pie with interior */
/* of wide cross hatch             */
if(vsf_style(dev_handle, WIDE_X ) < 0 )
   error_handler();
```

```
/* output pie slice ( mantle ) */
if(v_pieslice(dev_handle,
              centerx,
              centery,
              pc24x,
              2700,
              1800) < 0 )
   error_handler();

/* draw outer core - pie with interior */
/* of medium cross hatch               */
/* outer core is cyan */
if(vsf_style(dev_handle, MED_X ) < 0 )
   error_handler();

/* output pie slice ( outer core ) */
if(v_pieslice(dev_handle,
              centerx,
              centery,
              pc12_5x,
              2700,
              1800) < 0 )
   error_handler();

/* draw inner core - pie with interior  */
/* of narrow cross hatch                */
/* outer core is magenta */
if(vsf_style(dev_handle, NARROW_X ) < 0 )
   error_handler();

/* output pie slice ( outer core ) */
if(v_pieslice(dev_handle,
              centerx,
              centery,
              pc05x,
              2700,
              1800) < 0 )
   error_handler();

/* draw earth section - solid pie */
if(vsf_interior(dev_handle, SOLID) < 0)
   error_handler();

centerx = pc60x;
centery = pc40y;
```

```
 radius = pc25x;
 strt_ang = 2700;
 end_ang = 3600;

 /* output earth section */
 if(v_pieslice(dev_handle,
                  centerx,
                  centery,
                  radius,
                  strt_ang,
                  end_ang) < 0 )
    error_handler();

 /*  draw lines to point to the sections */

 xy[0] = pc60x;
 xy[1] = pc65y;
 xy[2] = pc80x;
 xy[3] = pc80y;
 if(v_pline(dev_handle, 2, xy) < 0 )
    error_handler();

 xy[0] = pc55x;
 xy[1] = pc55y;
 xy[2] = pc80x;
 xy[3] = pc65y;
 if(v_pline(dev_handle, 2, xy) < 0 )
    error_handler();

 xy[0] = pc49x;
 xy[1] = pc54y;
 xy[2] = pc10x;
 xy[3] = pc65y;
 if(v_pline(dev_handle, 2, xy) < 0 )
    error_handler();

/* set alignment */
if( vst_alignment(dev_handle,
                  CENTER,
                  TOP,
                  &align,
                  &align ) < 0 )
   error_handler();
 if(v_gtext(
       dev_handle,
       pc50x,
       yaxis,
       "EXAMPLE - USE OF 'PIE' FUNCTION"
     ) < 0 )
   error_handler();

 /* label sections */
/* set alignment */
if( vst_alignment(dev_handle,
                  LEFT,
                  BOTTOM,
                  &align,
                  &align ) < 0 )
  error_handler();

 if(v_gtext(dev_handle,
             pc75x,
             pc80y,
             "Mantle" ) < 0 )
   error_handler();

 if(v_gtext(dev_handle,
             pc73x,
             pc65y,
             "Outer core" ) < 0 )
   error_handler();

 if(v_gtext(dev_handle,
             pc02x,
             pc65y,
             "Inner core" ) < 0 )
   error_handler();

 if(v_gtext(dev_handle,
             pc02x,
             pc15y,
             "EARTH" ) < 0 )
   error_handler();

 if(v_gtext(dev_handle,
             pc02x,
             pc15y - chgt,
             "CROSS-SECTION" ) < 0 )
   error_handler();
```

```
                /* pause to view the frame */
                wait_kybrd();
             }
             else
               printf(
               "Pie slices not supported in this device\n\r"
               );
          }
          else
             printf(
             "No GDP's available on this device\n\r"
             );

          /* close the workstation */
          if (v_enter_cur(dev_handle) < 0 )
             error_handler();
          if (v_clswk(dev_handle) < 0 )
             error_handler();
       }
       else
         error_handler();

    }


    /****************************************/
    /*                                      */
             pie_slice(x, y)
    /*                                      */
    /****************************************/
    short x, y;
    {
       extern short vsf_color(), v_pieslice(), v_gtext();
    #define NUMBER_SLICES 8
    #define SWEEP (3600 / NUMBER_SLICES)
    #define PI 3.1416

       short centerx, centery, sangle, eangle, radius, i;
       short distance, align, label_x, label_y;
       double amplitude;
       extern short dev_handle, max_index, xaxis;
       extern double sin(), cos();
```

```
   /* define a label type */
   typedef struct {
                   short horz;
                   short vert;
                   char *label;
                  }  LABEL;

static LABEL label_list[] = {

                       {LEFT, BOTTOM, "Nar. Diag."},
                       {LEFT, BOTTOM, "Med. Diag."},
                       {RIGHT,BOTTOM, "Wide Diag."},
                       {RIGHT,BOTTOM, "Nar. Cross."},
                       {RIGHT,TOP   , "Med. Cross."},
                       {RIGHT,TOP   , "Wide Cross."},
                       {LEFT, TOP   , "Hollow"},
                       {LEFT, TOP   , "Solid"  }
             };

   /* set the size of pie slices */
   sangle = 0;
   eangle = SWEEP - 1;
   radius = pc25x;
   distance = pc17x;
   centerx = x;
   centery = y;

   if(vsf_interior(dev_handle, HATCH) < 0)
      error_handler();

   for( i = 0 ; i < NUMBER_SLICES ; i++){

      /* The first six pie slices are filled */
      /* with hatch interiors.               */
      /* The last two pie slices are hollow  */
      /* and solid interiors.                */

      if ( i < 6 ){ /* fill interior is hatch */
         if(vsf_style(dev_handle, i + 1) < 0 )
            error_handler();
      }
      else{ /* fill interior is solid or hollow */
         if(vsf_interior(dev_handle, i % 6) < 0)
            error_handler();
      }
      /* output pie slice */
      if(v_pieslice(dev_handle,
                    centerx,
                    centery,
                    radius,
                    sangle,
                    eangle) < 0 )
             error_handler();

      sangle += SWEEP;
      eangle += SWEEP;
   }

   /* set the size of pie slices */
   sangle = 0;
   eangle = SWEEP - 1;

   /* label the pie */
   for( i = 0 ; i < NUMBER_SLICES ; i++){

       /* label slice */
       /* set alignment */
       if( vst_alignment(dev_handle,
                         label_list[i].horz,
                         label_list[i].vert,
                         &align,
                         &align ) < 0 )
          error_handler();

       /* set label position by using  */
       /* polar coordinate conversion  */
       amplitude =
          (double)(sangle + (SWEEP / 2)) *
          (2.0 * PI) / 3600.0 ;

       label_x =
          (short)((double)distance * cos(amplitude));
       label_y =
          (short)((double)distance * sin(amplitude));

       /* output text label for pie interior */
       if(v_gtext(dev_handle,
                  centerx + label_x,
                  centery + label_y,
                  label_list[i].label) < 0)
          error_handler();
```

```
         sangle += SWEEP;
         eangle += SWEEP;
      }

      /* reset alignment */
      if( vst_alignment(dev_handle,
                        LEFT,
                        BOTTOM,
                        &align,
                        &align ) < 0 )
            error_handler();

   }



   /****************************************/
   /*                                      */
                wait_kybrd()
   /*                                      */
   /****************************************/
   {
     extern short vrq_string();
     short xy[2];
     char ch;

     if(vrq_string(dev_handle, 1, NO_ECHO, xy, &ch) < 0 )
        error_handler();
   }


   /****************************************/
   /*                                      */
              error_handler()
   /*                                      */
   /****************************************/
   {
      extern short vq_error();
      printf(" GDT error, number = %d\n\r", vq_error());
   }
```

Example 4—Polyline Function

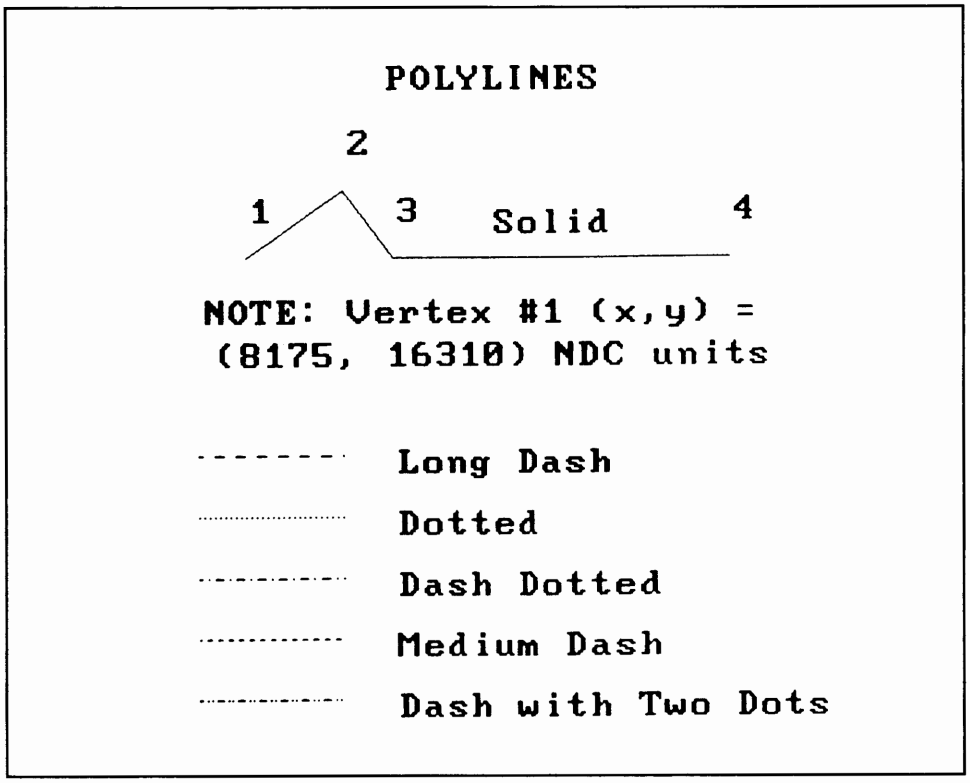


Figure B-8. Polyline Function, Part One

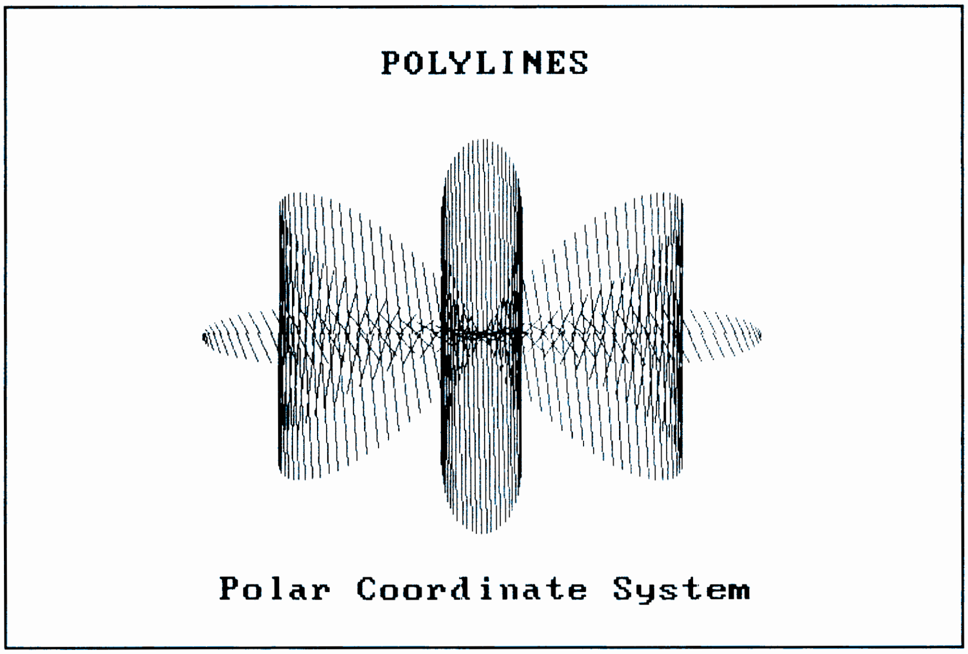


Figure B-9. Polyline Function, Part Two

```
#include <stdio.h>

#define NO_ECHO 0
#define ECHO 1

#define BLACK 0
#define WHITE 1
#define CYAN  6
#define MAGENTA 7

#define SOLID 1

#define CENTER 1
#define LEFT 0
#define RIGHT 2
#define TOP 2
#define BOTTOM 0

short dev_handle;
```

```
main()
/*********************************************/
/*                                           */
/*                                           */
/*  Example Program - Use of Polyline Function  */
/*                                           */
/*                                           */
/*********************************************/
{

extern short v_opnwk(), v_clrwk(), vst_height();
extern short vsl_type(), vst_color();
extern short v_gtext(), v_pline(), vsl_color();
extern short v_enter_cur(), v_clswk();

extern double cos(), sin();

short savary[66];
static short savin[]={ 1,
                       1,
                       1,
                       3,
                       1,
                       1,
                       1,
                       0,
                       0,
                       1,
                       1,
                       'D','I','S','P','L','A','Y',' '};
static char *lstyles[] = { "",
                           "Solid",
                           "Long Dash",
                           "Dotted",
                           "Dash Dotted",
                           "Medium Dash",
                           "Dash with Two Dots" };
short align, xwid, cwid, chgt, tx, ty;

short pc08y, pc10y, pc60y;
short pc63y, pc70y, pc80y, pc95y;
short pc02y, pc20x, pc30x;
short pc25x, pc45x, pc75x;
short pc35x, pc40x, pc50x, pc65x;
short tmp, ltype, xhalf, yhalf, angle;
float factor2, radians, R;
short xy[12], color, max_index;
short xaxis, yaxis, i;
char label[81];


   /* open the workstation */
   if(v_opnwk( savin, &dev_handle, savary ) >= 0 ){

      /* if # of line widths = 0, close workstation */
      if( savary[6] ){

         /*  savary[51] = max. NDC space - x axis */
         /*  savary[52] = max. NDC space - y axis */
         xaxis = savary[51];
         yaxis = savary[52];

         /* set up tags as %'s of max. x and y axes */
         pc02y = (yaxis / 100) * 2;
         pc08y = (yaxis / 100) * 8;
         pc10y = (yaxis / 100) * 10;
         pc60y = (yaxis / 100) * 60;
         pc63y = (yaxis / 100) * 63;
         pc70y = (yaxis / 100) * 70;
         pc80y = (yaxis / 100) * 80;
         pc95y = (yaxis / 100) * 95;

         pc20x = (xaxis / 100) * 20;
         pc25x = (xaxis / 100) * 25;
         pc30x = (xaxis / 100) * 30;
         pc35x = (xaxis / 100) * 35;
         pc40x = (xaxis / 100) * 40;
         pc45x = (xaxis / 100) * 45;
         pc50x = (xaxis / 100) * 50;
         pc65x = (xaxis / 100) * 65;
         pc75x = (xaxis / 100) * 75;

         /* 1rst drawing = polyline primitive */
         /* clear workstation */
         if(v_clrwk(dev_handle) < 0 )
            error_handler();
```

```
/* set text height to 1600 NDC units */
if(vst_height(dev_handle,
              1600,
              &xwid,
              &cwid,
              &chgt ) < 0 )
   error_handler();

/* set line type = solid */
if(vsl_type(dev_handle, SOLID) < 0 )
   error_handler();

/* set line color */
if(vst_color(dev_handle, MAGENTA ) < 0 )
   error_handler();

if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &align,
                 &align ) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           pc50x,
           yaxis,
           "POLYLINES") < 0)
   error_handler();

/* draw solid 4 point polyline */
xy[0] = pc25x;
xy[1] = pc70y;
xy[2] = pc35x;
xy[3] = pc80y;
xy[4] = pc40x;
xy[5] = pc70y;
xy[6] = pc75x;
xy[7] = pc70y;
if(v_pline(dev_handle, 4, xy) < 0 )
   error_handler();

/* set line color */
if(vsl_color(dev_handle, CYAN ) < 0 )
   error_handler();
```

```
if(vst_alignment(dev_handle,
                 LEFT,
                 CENTER,
                 &align,
                 &align ) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           pc50x,
           pc70y + chgt,
           "Solid") < 0)
   error_handler();

/* set up increment on y axis */
/* label vertices of solid polyline */
for ( i = 0 ; i < 7 ; i+=2 ){
   tmp = (i/2) + 1 ;
   sprintf( label, "%d", tmp);
   tx = xy[i] ;
   ty = xy[i + 1] + pc08y;
   if(v_gtext(dev_handle, tx, ty, label) < 0 )
      error_handler();
}


/* set text color */
if(vst_color(dev_handle, MAGENTA ) < 0 )
   error_handler();

if(vst_alignment(dev_handle,
                 CENTER,
                 CENTER,
                 &align,
                 &align ) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           pc50x,
           pc63y,
           "NOTE: Vertex #1 (x,y) = ") < 0)
   error_handler();

sprintf( label,
         "(%d, %d) NDC units",
         xy[0],
         xy[1]);
```

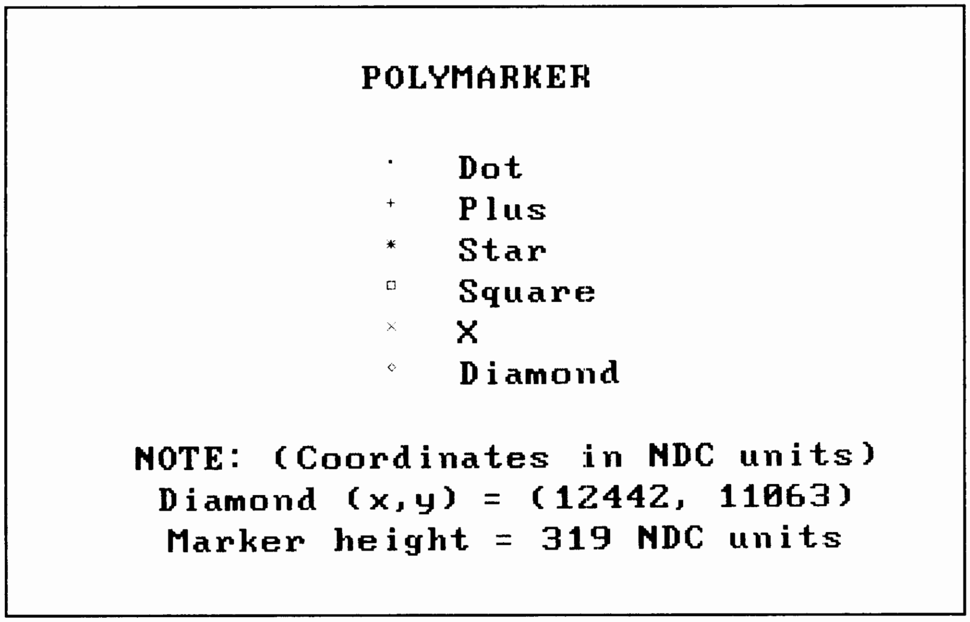
```
if(v_gtext(dev_handle,
           pc50x,
           pc63y - chgt,
           label) < 0)
   error_handler();

/* set maximum device color index */
max_index = savary[39] - 1;

if(vst_alignment(dev_handle,
                 LEFT,
                 CENTER,
                 &align,
                 &align ) < 0 )
   error_handler();

/* draw remaining line styles */
xy[0] = pc20x;
xy[2] = pc35x;
tx = pc40x;
color = WHITE;

for ( ltype = 2 ; ltype  < 7 ; ltype++ ){
   xy[3] = xy[1] =
      (short)
      ((0.5 - ((float)(ltype - 1) * 0.09)) *
      (float)yaxis);


   /* set text color */
   if(vst_color(dev_handle, color) < 0 )
      error_handler();

   if(v_gtext(dev_handle,
              tx,
              xy[3],
              lstyles[ltype] ) < 0)

      error_handler();

   /* set line type */
   if(vsl_type(dev_handle, ltype) < 0 )
      error_handler();
   /* set line color */
   if(vsl_color(dev_handle, color) < 0 )
      error_handler();

   /*  output 2 point polyline */
   if(v_pline(dev_handle, 2, xy) < 0)
      error_handler();

   if ( ++color > max_index ) color = WHITE;
}

wait_kybrd();
if(v_clrwk(dev_handle) < 0 )
   error_handler();

/*********************************************/
/*                                           */
/*    Draw graph using polylines and         */
/*    polar coordinates (distance, angle).   */
/*                                           */
/*********************************************/

/* set text color */
if(vst_color(dev_handle, MAGENTA ) < 0 )
   error_handler();

if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &align,
                 &align ) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           pc50x,
           pc95y,
           "POLYLINES") < 0)
   error_handler();


factor2 = 3.14 / 180.0;
xhalf = xaxis / 2;
yhalf = yaxis / 2;
color = WHITE;
```

```
/* set line color */
if(vsl_color(dev_handle, color) < 0 )
   error_handler();

 /* set line type */
 if(vsl_type(dev_handle, SOLID) < 0 )
    error_handler();

 /* use the equation ' R = 6cos4A ' */
 /* where A = angle and            */
 /* R = distance from origin       */
 /* calculate x coord. : x = RcosA */
 /* calculate y coord. : y = RsinA */
 radians = factor2;
 R =
    (6.0 *
    (float)cos((double)(4.0 * radians))) /
     10.0;

 xy[0] =
    (short)(R *
           (float)cos((double)radians) *
           (float)xaxis);
 xy[0] = (xy[0]/2) + xhalf;

 xy[1] =
    (short)(R *
           (float)sin((double)radians) *
           (float)yaxis);
 xy[1] = (xy[1]/2) + yhalf;

 /* output graph in polar coord. system  */
 for ( angle = 2 ; angle <= 360 ; angle++ ){

     radians = (float)angle * factor2;
     R =

        (6.0 *
        (float)cos((double)(4.0 * radians))) /
        10.0;

     xy[2] =
        (short)(R *
               (float)cos((double)radians) *
               (float)xaxis);
```

```
                xy[2] = (xy[2]/2) + xhalf;
                xy[3] =
                   (short)(R *
                          (float)sin((double)radians) *
                          (float)yaxis);
                xy[3] = (xy[3]/2) + yhalf;

                if(v_pline(dev_handle,2,xy) < 0 )
                   error_handler();

                /* set next line color */
                if ( ++color > max_index ) color = WHITE ;
                if(vsl_color(dev_handle, color ) < 0 )
                   error_handler();
                xy[0] = xy[2];
            }

            /* set text color */
            if(vst_color(dev_handle, CYAN) < 0 )
               error_handler();

            /* label graph */
            if(vst_alignment(dev_handle,
                             CENTER,
                             BOTTOM,
                             &align,
                             &align ) < 0 )
               error_handler();

            if(v_gtext(dev_handle,
                       pc50x,
                       pc10y,
                       "Polar Coordinate System") < 0)
               error_handler();

            wait_kybrd();
        }
        else{
           printf("Device not capable of graphics\n\r");
        }
```

```
      /* close the workstation */
      if (v_enter_cur(dev_handle) < 0 )
         error_handler();
      if (v_clswk(dev_handle) < 0 )
         error_handler();
   }
   else
     error_handler();

}

/*****************************************/
/*                                       */
             wait_kybrd()
/*                                       */
/*****************************************/
{
  extern short vrq_string();
  short xy[2];
  char ch;

  if(vrq_string(dev_handle, 1, NO_ECHO, xy, &ch) < 0 )
     error_handler();
}

/*****************************************/
/*                                       */
           error_handler()
/*                                       */
/*****************************************/
{
   extern short vq_error();
   printf("GDT error, number = %d\n\r", vq_error());
}
```

Example 5 – Polymarker Function



```
              POLYMARKER

            ·      Dot
            +      Plus
            *      Star
            □      Square
            ×      X
            ◇      Diamond

NOTE: (Coordinates in NDC units)
 Diamond (x,y) = (12442, 11063)
  Marker height = 319 NDC units
```

Figure B-10. Polymarker Function, Part One

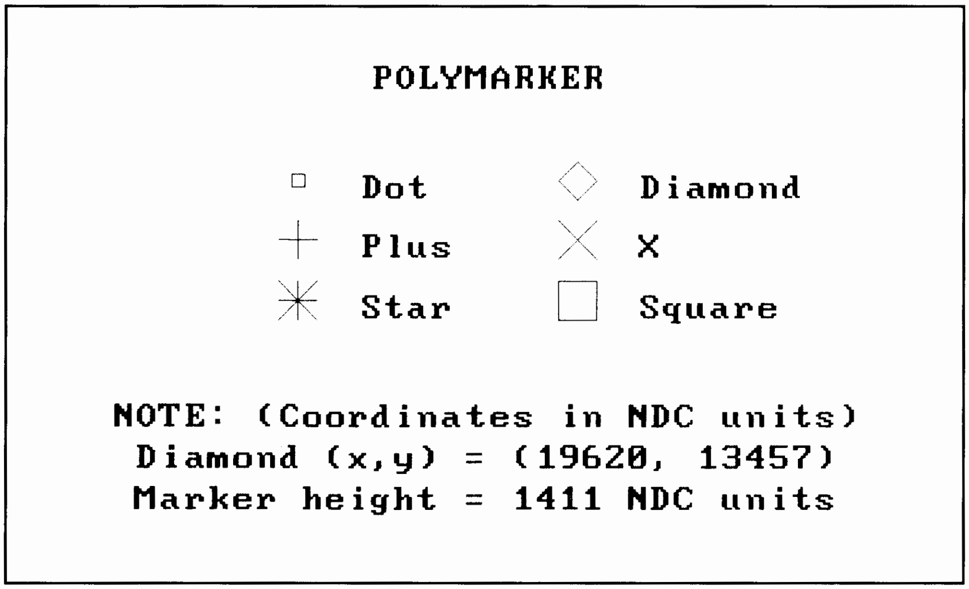

POLYMARKER
Dot
Diamond
Plus
X
Star
Square
NOTE: (Coordinates in NDC units)
Diamond (x,y) = (19620, 13457)
Marker height = 1411 NDC units

Figure B-11. Polymarker Function, Part Two

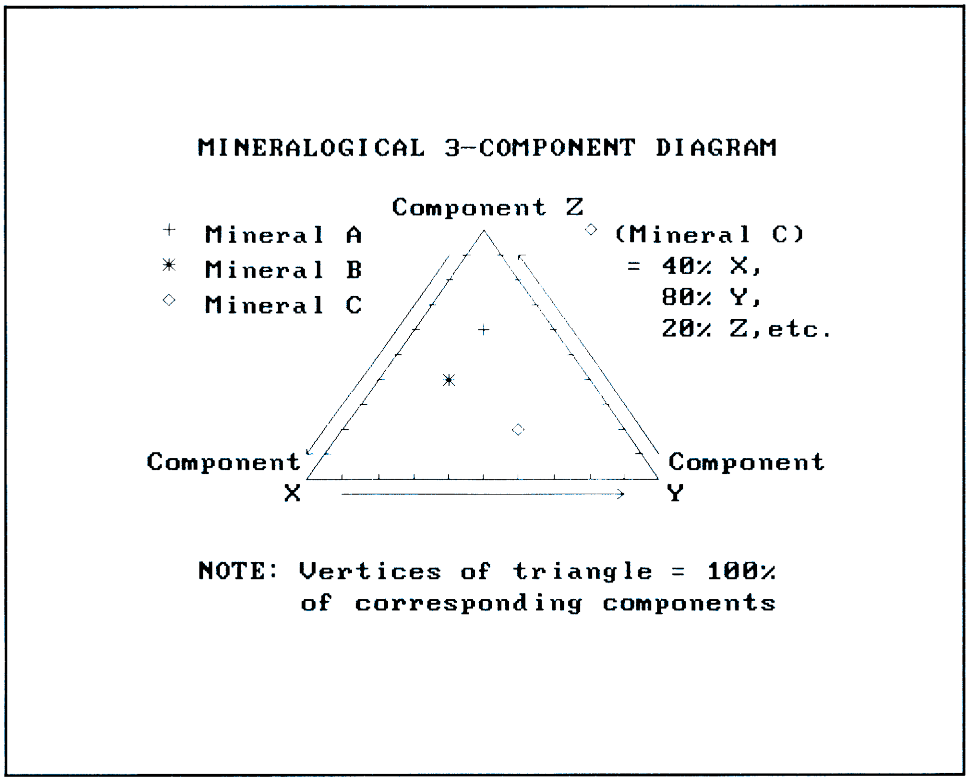


Figure B-12. Polymarker Function, Part Three

```
#include <stdio.h>

#define NO_ECHO 0
#define ECHO 1

#define DIAMOND_MARKER 6
#define STAR_MARKER 3
#define CROSS_MARKER 2

#define LEFT 0
#define RIGHT 2
#define CENTER 1
#define BOTTOM 0
#define TOP 2

#define BLACK 0
```

```
#define WHITE 1
#define CYAN  6
#define MAGENTA 7

#define SOLID 1


short dev_handle;

main()
/***********************************************/
/*                                             */
/*                                             */
/* Example Program - Use of Polymarker Function */
/*                                             */
/*                                             */
/***********************************************/
{

extern short v_opnwk(), v_clrwk(), vst_height();
extern short vsm_height(), v_gtext();
extern short vsm_type(), v_pmarker(), vsm_color();
extern short vst_color(), v_fillarea();
extern short vst_alignment(), v_pline(), vsl_color();
extern short v_enter_cur(), v_clswk();

short savary[66];
static short savin[]={ 1,
                       1,
                       1,
                       3,
                       1,
                       1,
                       1,
                       0,
                       0,
                       1,
                       1,
                      'D','I','S','P','L','A','Y',' '};

static char *mark_title[]  = {"Dot",
                               "Plus",
                               "Star",
                               "Square",
                               "X",
                               "Diamond" };
```

```
short xwid, cwid, chgt, tx, ty;


short xy[12], color, max_index;
short deltay , xaxis, yaxis, i, tmph, tmpv;
char label[81];

short thgt, mtyp, imhgt;
short pc02_5x;
short pc01x, pc02x, pc05x, pc07x, pc10x;
short pc15x, pc20x, pc23x, pc24x, pc25x;
short pc26x, pc30x, pc38x, pc40x, pc45x, pc47x;
short pc50x, pc55x, pc56x, pc60x, pc65x, pc68x;
short pc69x, pc70x, pc75x, pc76x, pc77x, pc80x;

short pc05y, pc10y, pc15y;
short pc26y, pc27y, pc28y, pc30y;
short pc31y, pc32y, pc35y, pc36y, pc40y;
short pc50y, pc55y, pc60y, pc65y, pc66y;
short pc73y, pc74y, pc75y, pc80y, pc83y;
short pc85y, pc88y, pc95y;

   /* open the workstation */
   if(v_opnwk( savin, &dev_handle, savary ) >= 0 ){

      /*  savary[51] = max. NDC space - x axis */
      /*  savary[52] = max. NDC space - y axis */
      xaxis = savary[51];
      yaxis = savary[52];

      /* set up tags as %'s of max. x and y axes */
      pc02_5x = (xaxis / 1000) * 25;
      pc01x = (xaxis / 100) * 1;
      pc02x = (xaxis / 100) * 2;
      pc05x = (xaxis / 100) * 5;
      pc07x = (xaxis / 100) * 7;
      pc10x = (xaxis / 100) * 10;
      pc15x = (xaxis / 100) * 15;
      pc20x = (xaxis / 100) * 20;
      pc23x = (xaxis / 100) * 23;
      pc24x = (xaxis / 100) * 24;
      pc25x = (xaxis / 100) * 25;
      pc26x = (xaxis / 100) * 26;
      pc30x = (xaxis / 100) * 30;
      pc38x = (xaxis / 100) * 38;
```

```
pc40x = (xaxis / 100) * 40;
pc45x = (xaxis / 100) * 45;
pc47x = (xaxis / 100) * 47;
pc50x = (xaxis / 100) * 50;
pc55x = (xaxis / 100) * 55;
pc56x = (xaxis / 100) * 56;
pc60x = (xaxis / 100) * 60;
pc65x = (xaxis / 100) * 65;
pc68x = (xaxis / 100) * 68;
pc69x = (xaxis / 100) * 69;
pc70x = (xaxis / 100) * 70;
pc75x = (xaxis / 100) * 75;
pc76x = (xaxis / 100) * 76;
pc77x = (xaxis / 100) * 77;
pc80x = (xaxis / 100) * 80;

pc05y = (yaxis / 100) * 5;
pc10y = (yaxis / 100) * 10;
pc15y = (yaxis / 100) * 15;
pc26y = (yaxis / 100) * 26;
pc27y = (yaxis / 100) * 27;
pc28y = (yaxis / 100) * 28;
pc30y = (yaxis / 100) * 30;
pc31y = (yaxis / 100) * 31;
pc32y = (yaxis / 100) * 32;
pc35y = (yaxis / 100) * 35;
pc36y = (yaxis / 100) * 36;
pc40y = (yaxis / 100) * 40;
pc50y = (yaxis / 100) * 50;
pc55y = (yaxis / 100) * 55;
pc60y = (yaxis / 100) * 60;
pc65y = (yaxis / 100) * 65;
pc66y = (yaxis / 100) * 66;
pc73y = (yaxis / 100) * 73;
pc74y = (yaxis / 100) * 74;
pc75y = (yaxis / 100) * 75;
pc80y = (yaxis / 100) * 80;
pc83y = (yaxis / 100) * 83;
pc85y = (yaxis / 100) * 85;
pc88y = (yaxis / 100) * 88;
pc95y = (yaxis / 100) * 95;

/* max. device color index */
max_index = savary[39] - 1;
```

```
/* show polymarkers with height = minimum */
if(v_clrwk(dev_handle) < 0 )
   error_handler();

/* set requested text height to 1600 NDC units */
if(vst_height(dev_handle,
              1600,
              &xwid,
              &cwid,
              &chgt ) < 0 )
   error_handler();

if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &tmph,
                 &tmpv ) < 0)

   error_handler();

/* set polymarker height to minimum */
imhgt = savary[64];
if(vsm_height(dev_handle, imhgt) < 0 )
   error_handler();

if(v_gtext(dev_handle,
           pc47x,
           pc95y,
           "POLYMARKER") < 0 )
   error_handler();

if(vst_alignment(dev_handle,
                 LEFT,
                 CENTER,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();


color = WHITE;
/* center the text and marker output */
tx = pc40x;
xy[0] = tx  - (imhgt * 2);
```

```
/* determine the amount to step down */
/* for each output line              */
deltay = imhgt < chgt ? chgt : imhgt * 2;
for ( i = 0; i < 6 ; i++){
   xy[1] = pc85y - ((i+1) * deltay);

   /* set the marker type */
   if(vsm_type(dev_handle, i + 1) < 0 )
      error_handler();

   /* output one marker */
   if(v_pmarker(dev_handle,1, xy) < 0 )
      error_handler();

   /* write text */
   if(v_gtext(dev_handle,
              tx + pc05x,
              xy[1],
              mark_title[i]) < 0 )
      error_handler();

   if( ++color > max_index ) color = WHITE;
   if(vsm_color(dev_handle, color ) < 0)
      error_handler();
   if(vst_color(dev_handle, color ) < 0)
      error_handler();
}

if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();

thgt = xy[1] - pc10y;
/* describe graphics drawing */
if(v_gtext(
      dev_handle,
      pc50x,
      thgt,
      "NOTE: (Coordinates in NDC units)") < 0)
   error_handler();
```

```
/* output position */
sprintf( label,
         "Diamond (x,y) = (%d, %d)",
         xy[0],
         xy[1]);
thgt -= chgt;
if(v_gtext(dev_handle, pc50x, thgt, label) < 0 )
   error_handler();
/* output marker height */
sprintf( label,
         "Marker height = %d NDC units",
         imhgt);

thgt -= chgt;
if(v_gtext(dev_handle, pc50x, thgt, label) < 0 )
   error_handler();

wait_kybrd();
if(v_clrwk(dev_handle) < 0 ) error_handler();

/* show polymarkers with height = maximum */
imhgt = savary[65];
if(vsm_height(dev_handle, imhgt) < 0 )
   error_handler();

/* set text color */
color = WHITE;
if(vst_color(dev_handle, color) < 0 )
   error_handler();

/* set marker color */
if(vsm_color(dev_handle, color) < 0 )
   error_handler();

/* output label */
if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();
```

```
if(v_gtext(dev_handle,
           pc50x,
           pc95y,
           "POLYMARKER") < 0)
   error_handler();

if(vst_alignment(dev_handle,
                 LEFT,
                 CENTER,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();

xy[1] = pc85y;
for ( i = 0 ; i < 3 ; i++){
   xy[0] = pc30x;
   xy[1] -= (imhgt + (imhgt /2));

   /* set polymarker type */
   if(vsm_type(dev_handle, i + 1) < 0)
      error_handler();

   /* output a single marker */
   if(v_pmarker(dev_handle, 1, xy) < 0 )
      error_handler();

   /* output label */
   tx = xy[0] + (imhgt + (imhgt/2));
   ty = xy[1];
   if(v_gtext(dev_handle,
              tx,
              ty,
              mark_title[i]) < 0 )
      error_handler();

   /* set marker color and text color */
   if( ++color > max_index ) color = WHITE;
   if(vsm_color(dev_handle, color) < 0)
      error_handler();
   if(vst_colcr(dev_handle, color) < 0)
      error_handler();

   xy[0] = pc60x;
   mtyp = 6 - i;
```

```
      /* set polymarker type */
      if(vsm_type(dev_handle, mtyp) < 0)
         error_handler();

      /* output a single marker */
      if(v_pmarker(dev_handle, 1, xy) < 0 )
         error_handler();

      /* output label */
      tx = xy[0] + (imhgt + (imhgt/2));
      if(v_gtext(dev_handle,
                 tx,
                 ty,
                 mark_title[5 - i]) < 0 )
         error_handler();

      /* set marker color and text color */
      if( ++color > max_index ) color = WHITE;
      if(vsm_color(dev_handle, color) < 0)
         error_handler();
      if(vst_color(dev_handle, color) < 0)
         error_handler();
   }

   /* describe graphics drawing */
   if(vst_color(dev_handle, CYAN ) < 0)
      error_handler();

   if(vst_alignment(dev_handle,
                    CENTER,
                    TOP,
                    &tmph,
                    &tmpv ) < 0)
      error_handler();

   thgt = xy[1] - pc10x;

   if(v_gtext(dev_handle,
         pc50x,
         thgt,
         "NOTE: (Coordinates in NDC units)") < 0 )
      error_handler();
```

```
/* output position */
sprintf( label,
         "Diamond (x,y) = (%d, %d)",
         xy[0],
         xy[1]);

thgt -= chgt;
if(v_gtext(dev_handle, pc50x, thgt, label) < 0 )
   error_handler();

/* output marker height */
sprintf( label,
         "Marker height = %d NDC units",
         imhgt);

thgt -= chgt;
if(v_gtext(dev_handle, pc50x, thgt, label) < 0 )
   error_handler();


wait_kybrd();
if(v_clrwk(dev_handle) < 0 )
   error_handler();

/*  show polymarkers use      */
/*  (Mineralogical diagram)   */


/* output title */
if(vst_alignment(dev_handle,
                 CENTER,
                 TOP,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();

if(v_gtext(
      dev_handle,
      pc50x,
      yaxis,
      "MINERALOGICAL 3-COMPONENT DIAGRAM") < 0 )
   error_handler();

/* draw triangle */
xy[0] = pc25x;
```

```
xy[1] = pc30y;
xy[2] = pc50x;
xy[3] = pc80y;
xy[4] = pc75x;
xy[5] = pc30y;
if(v_fillarea(dev_handle, 3, xy) < 0)
   error_handler();


/* label corners of the triangle */
if(vst_alignment(dev_handle,
                 LEFT,
                 BOTTOM,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();

if(v_gtext(dev_handle,
           pc76x,
           pc32y,
           "Component") < 0)
   error_handler();

if(v_gtext(dev_handle,
           pc76x,
           pc32y - chgt,
           "Y") < 0)
   error_handler();

if(vst_alignment(dev_handle,
                 RIGHT,
                 BOTTOM,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();

if(v_gtext(dev_handle,
           pc24x,
           pc32y
           , "Component") < 0)
   error_handler();
```

```
if(v_gtext(dev_handle,
           pc24x,
           pc32y - chgt,
           "X") < 0)
   error_handler();

if(vst_alignment(dev_handle,
                 CENTER,
                 BOTTOM,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();

if(v_gtext(dev_handle,
           pc50x,
           pc83y,
           "Component Z") < 0)
   error_handler();

if(vst_alignment(dev_handle,
                 LEFT,
                 CENTER,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();

/*  place percent lines on triangle */
xy[0] = pc75x;
xy[1] = pc30y;

for( i = 1 ; i < 10 ; i++){
   xy[0] -= pc02_5x;
   xy[1] += pc05y;
   xy[2] = xy[0] - pc01x;
   xy[3] = xy[1];
   if(v_pline(dev_handle, 2, xy) < 0)
      error_handler();
}

xy[0] = pc25x;
xy[1] = pc30y;
for( i = 1 ; i < 10 ; i++){
   xy[0] += pc02_5x;
   xy[1] += pc05y;
   xy[2] = xy[0] + pc01x;
```

```
      xy[3] = xy[1];
      if(v_pline(dev_handle, 2, xy) < 0)
         error_handler();
   }

   xy[0] = pc25x;
   xy[1] = pc30y;
   xy[3] = pc31y;

   for( i = 1 ; i < 10 ; i++){
      xy[0] += pc05x;
      xy[2] = xy[0];
      if(v_pline(dev_handle, 2, xy) < 0)
         error_handler();
   }

   xy[0] = pc30x;
   xy[1] = pc27y;
   xy[2] = pc70x;
   xy[3] = pc27y;
   if(v_pline(dev_handle, 2, xy) < 0)
      error_handler();

   xy[0] = pc69x;
   xy[1] = pc28y;
   if(v_pline(dev_handle, 2, xy) < 0)
      error_handler();

   xy[1] = pc26y;
   if(v_pline(dev_handle, 2, xy) < 0)
      error_handler();

   xy[0] = pc75x;
   xy[1] = pc35y;
   xy[2] = pc55x;
   xy[3] = pc75y;
   if(v_pline(dev_handle, 2, xy) < 0)
      error_handler();

   xy[0] = xy[2];
   xy[1] = pc74y;
   if(v_pline(dev_handle, 2, xy) < 0)
      error_handler();
```

```
xy[0] = pc56x;
xy[1] = xy[3];
if(v_pline(dev_handle, 2, xy) < 0)
   error_handler();

xy[0] = pc45x;
xy[1] = pc75y;
xy[2] = pc25x;
xy[3] = pc35y;
if(v_pline(dev_handle, 2, xy) < 0)
   error_handler();

xy[0] = pc26x;
xy[1] = xy[3];
if(v_pline(dev_handle, 2, xy) < 0)
   error_handler();

xy[0] = xy[2];
xy[1] = pc36y;
if(v_pline(dev_handle, 2, xy) < 0)
   error_handler();


/* set marker height to 800 NDC units */
if(vsm_height(dev_handle, 800) < 0 )
   error_handler();

/* output polymarkers on diagram */

/* output a '+' marker */
if(vsm_type(dev_handle, CROSS_MARKER) < 0 )
   error_handler();

/* set marker color */
if(vsm_color(dev_handle, WHITE) < 0)
   error_handler();

/* output the marker */
xy[0] = pc50x;
xy[1] = pc60y;
if(v_pmarker(dev_handle, 1, xy) < 0 )
   error_handler();
```

```
/* set text color */
if(vst_color(dev_handle, WHITE) < 0)
   error_handler();

/* output marker */
xy[0] = pc05x;
xy[1] = pc80y;
if(v_pmarker(dev_handle, 1, xy) < 0 )
   error_handler();

/* output label */
if(v_gtext(dev_handle,
           pc10x,
           xy[1],
           "Mineral A") < 0)
   error_handler();

/* output a '*' marker */
if(vsm_type(dev_handle, STAR_MARKER) < 0 )
   error_handler();


/* output marker */
xy[0] = pc45x;
xy[1] = pc50y;
if(v_pmarker(dev_handle, 1, xy) < 0 )
   error_handler();

/* output marker */
xy[0] = pc05x;
xy[1] = pc73y;
if(v_pmarker(dev_handle, 1, xy) < 0 )
   error_handler();

/* output label */
if(v_gtext(dev_handle,
           pc10x,
           xy[1],
           "Mineral B") < 0)
   error_handler();


/* output a diamond marker */
if(vsm_type(dev_handle, DIAMOND_MARKER) < 0 )
   error_handler();
```

```
/* output marker */
xy[0] = pc55x;
xy[1] = pc40y;
if(v_pmarker(dev_handle, 1, xy) < 0 )
   error_handler();

/* output marker */
xy[0] = pc05x;
xy[1] = pc66y;
if(v_pmarker(dev_handle, 1, xy) < 0 )
   error_handler();

/* output label */
if(v_gtext(dev_handle,
           pc10x,
           xy[1],
           "Mineral C") < 0)
   error_handler();


/* explain the polymarker positioning */


xy[0] = pc65x;
xy[1] = pc80y;
if(v_pmarker(dev_handle, 1, xy) < 0 )
   error_handler();

/* output label */
if(v_gtext(dev_handle,
           pc68x,
           xy[1],
           "(Mineral C)") < 0)
   error_handler();

xy[1] -= chgt;
if(v_gtext(dev_handle,
           pc70x,
           xy[1],
           "= 40% X,") < 0)
   error_handler();
```

```
xy[1] -= chgt;
if(v_gtext(dev_handle,
           pc70x,
           xy[1],
           "  80% Y,") < 0)
   error_handler();

xy[1] -= chgt;
if(v_gtext(dev_handle,
           pc70x,
           xy[1],
           "  20% Z,etc.") < 0)
   error_handler();

thgt = pc10y;

if(vst_alignment(dev_handle,
                 CENTER,
                 BOTTOM,
                 &tmph,
                 &tmpv ) < 0)
   error_handler();


if(v_gtext(
      dev_handle,
      pc50x,
      thgt,
      "NOTE: Vertices of triangle = 100%") < 0)
   error_handler();

thgt -= chgt;
if(v_gtext(
      dev_handle,
      pc50x,
      thgt,
      "      of corresponding components") < 0)
   error_handler();

wait_kybrd();
```

```
         /* close the workstation */
         if (v_enter_cur(dev_handle) < 0 )
            error_handler();
         if (v_clswk(dev_handle) < 0 )
            error_handler();
      }
      else
        error_handler();

   }

   /****************************************/
   /*                                      */
               wait_kybrd()
   /*                                      */
   /****************************************/
   {
     extern short vrq_string();
     short xy[2];
     char ch;

     if(vrq_string(dev_handle, 1, NO_ECHO, xy, &ch) < 0 )
        error_handler();
   }


   /****************************************/
   /*                                      */
             error_handler()
   /*                                      */
   /****************************************/
   {
      extern short vq_error();
      printf("GDT error, number = %d\n\r", vq_error());
   }
```

Appendix C. Graphics Drivers

This appendix contains information about the device drivers for those peripheral devices that are supported by the Graphics Development Toolkit. It describes the capabilities and limitations of the devices, the index numbers used to select functions (such as marker types, line styles, and colors), and other special information.

The Graphics Development Toolkit supports two program modes: Graphics and Cursor Text. It is important to remember that only one of these modes can be active at one time. By default, the "Open Workstation" routine places a program in Graphics mode. The program can be switched to Cursor Text mode by calling the "Enter Cursor Text Mode" routine. A program must be switched back into Graphics mode by invoking the "Exit Cursor Text Mode" routine.

Four of the device drivers described in this appendix support the operation of the IBM RT PC Mouse, as well as cursor movement keys on a keyboard. These device drivers are:

- IBM Advanced Monochrome Graphics Display
- IBM Advanced Color Graphics Display
- IBM Extended Monochrome Graphics Display
- IBM Enhanced Graphics Adapter.

The following table lists the device drivers described in this appendix and gives the page number where each description begins.

Graphics Development Toolkit Device Drivers	Page
IBM 3812 Printer	C-6
IBM 4201 Printer	C-23
IBM 5152 Graphics Printer	C-31
IBM 5182 Color Printer	C-39
IBM 6180 Color Plotter	C-48
IBM 7371 Color Plotter	C-56
IBM 7372 Color Plotter	C-63
IBM 7374, 7375-1, 7375-2 Plotters	C-70
IBM Advanced Monochrome Graphics Display	C-80
IBM Advanced Color Graphics Display	C-88
IBM Extended Monochrome Graphics Display	C-96
IBM Enhanced Graphics Adapter	C-104
IBM Virtual Device Metafile (VDM) Driver	C-114
IBM RT PC Grafstation Driver	C-120

Device Driver Management

A system can have many graphics input and output peripherals attached, each requiring a separate device driver to interface to the system. The Graphics Development Toolkit receives requests for graphics devices from the application program, and ensures that the proper device driver is loaded in memory. When required, a device driver is loaded dynamically.

Because capabilities vary among graphics devices, the Graphics Development Toolkit emulates certain device capabilities. The Toolkit also provides feedback to the caller about the actual capabilities of the currently open device.

The Toolkit Routines

The Graphics Development Toolkit provides routines for:

- Device driver management
- Coordinate transformation
- Emulation of certain graphics primitives
- Error reporting.

Installation Procedures

For a step-by-step procedure to install the Graphics Development Toolkit on your system, refer to Appendix A, "Installing the Graphics Development Toolkit."

Incorporating Graphics Into Application Programs

You can incorporate graphics operations into your application programs by including calls to the Graphics Development Toolkit routines in your source code. The operations and parameters for the Graphics Development Toolkit graphics routines are detailed in Chapter 3, "Toolkit Routines."

The code that implements the interface between the Graphics Development Toolkit and your programming language is contained in a language library that is included on the distribution diskette. After your program is compiled, it must be linked with this language library and any other required modules and libraries. To link your program, follow the procedure shown in the Language Reference Booklet that supports the programming language you are using. In general, the language library should be linked after your application code and before any language support libraries.

The executable program module is executed and debugged in the usual manner with the tools provided by the operating system. Since the Graphics Development Toolkit routines always return status information to the calling program rather than displaying error messages, we recommend that you include a very simple status-checking routine in your program to be called after each graphics routine. The purpose of this status-checking routine is to display any error codes that arise before you have finished debugging your application's error handling routines.

Note: Terminating an application program without using Close Workstation may cause unpredictable results. Refer to the hardware documentation of the individual device to reset the device.

Logical Device Name

The logical device name is a user-selectable string, up to eight characters long, that represents the workstation identifier for a device. This string must be included in the call to the Open Workstation routine that initializes the device. In addition, this string must be exported to the system environment to configure the device.

Refer to Appendix A, "Installing the Graphics Development Toolkit", for information about exporting parameters to the system environment.

Communications

There are some devices that require a communications interface between the device and the system. For additional information about these drivers, refer to the *IBM RT PC User Setup Guide* or individual device manuals.

How Device Drivers are Described

Each description of a device driver begins with the device driver's **Filename.** This is a single word that represents the only legal name for the device driver file. The remainder of each description is organized into three information categories:

Features Supported

This information category lists and describes the features that are supported by the device driver. Each feature is shown under a sub-heading that follows the general heading of "Features Supported."

Device Specific Information

This information category describes any special operating considerations that are specific to a device or device driver.

Routine Summary

This information category lists the name of each Graphics Development Toolkit routine that is supported by the device driver.

IBM 3812 Printer

Filename: vdi3812

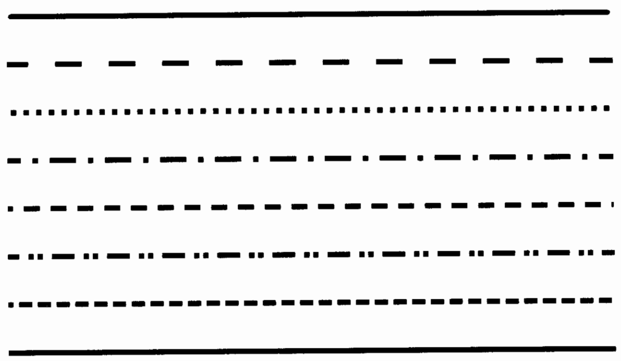
Features Supported

The following text describes each of the IBM 3812 Printer features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM 3812 Printer with one of eight line styles, selected with style indexes 1 thru 8:

1=Solid (round ends)
2=Long Dash
3=Dotted
4=Dash Dotted
5=Medium Dash
6=Dash With Two Dots
7=Short Dash
8=Solid (square ends)



Graphics Markers

The IBM 3812 Printer supports six Graphics Marker types, selected with type indexes 1 thru 6:

1=Dot	■
2=Cross	+
3=Star	✱
4=Square	□
5=X	×
6=Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following tables show the NDC units for the five size indexes for eight possible combinations of printer options (legal or letter, portrait or landscape, and margin or no margin). Refer to the later discussion of "Device Specific Information" for a description of the printer options.

Graphics Marker Sizes
Letter Portrait Non-Margin

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	236	236
2	460	460
3	683	683
4	907	907
5	1130	1130

Graphics Marker Sizes Letter Landscape Non-Margin		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	311	402
2	609	788
3	907	1173
4	1204	1559
5	1502	1944

Graphics Marker Sizes Legal Portrait Non-Margin		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	186	186
2	361	361
3	537	537
4	712	712
5	888	888

Graphics Marker Sizes Legal Landscape Non-Margin		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	303	498
2	595	980
3	888	1462
4	1181	1944
5	1473	2426

Graphics Marker Sizes Letter Portrait Margin		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	248	248
2	482	482
3	716	716
4	950	950
5	1184	1184

Graphics Marker Sizes Letter Landscape Margin		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	326	427
2	638	837
3	950	1246
4	1262	1656
5	1574	2066

Graphics Marker Sizes Legal Portrait Margin		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	193	193
2	375	375
3	557	557
4	739	739
5	921	921

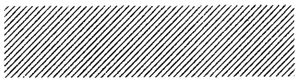
Graphics Marker Sizes Legal Landscape Margin		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	314	530
2	617	1042
3	921	1554
4	1224	2066
5	1528	2578

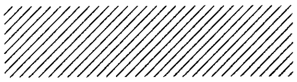
Graphics Text

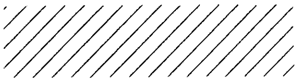
The IBM 3812 Printer supports continuous character scaling for the size of Graphics Text. This text can be rotated on 0, 90, 180, and 270 degree baselines.

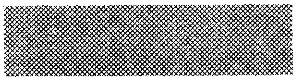
Filled Areas

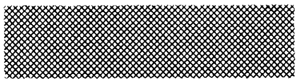
Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. There are six Pattern interior styles (index numbers one thru six) that are mapped to six Hatch interior styles. However, these six and an additional twenty-nine (thirty-five total) Hatch patterns can be used for GDPs only, as follows:

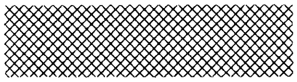
1=Right narrow diagonal 

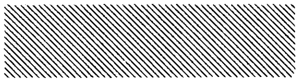
2=Right medium diagonal 

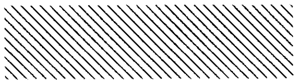
3=Right wide diagonal 

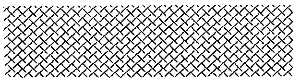
4=Narrow diagonal crosshatch 

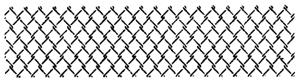
5=Medium diagonal crosshatch 

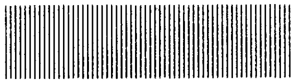
6=Wide diagonal crosshatch 

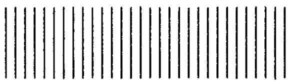
7=Left narrow diagonal 

8=Left medium diagonal 

9=Narrow fencing 

10=Wide fencing 

11=Narrow vertical lines 

12=Medium vertical lines 

13 = Narrow horizontal lines

14 = Medium horizontal

15 = Horz/Vert crosshatch 1

16 = Horz/Vert crosshatch 2

17 = Horz/Vert crosshatch 3

18 = Horz/Vert crosshatch 4

19 = Horz/Vert crosshatch 5

20 = Horz/Vert crosshatch 6

21 = Horz/Vert crosshatch 7

22 = Horz/Vert crosshatch 8

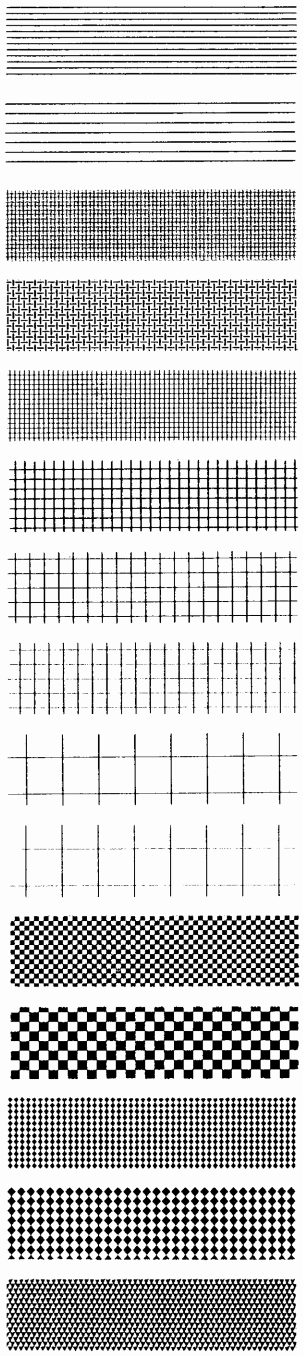
23 = Little checkerboard

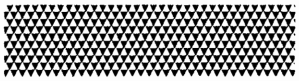
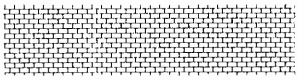
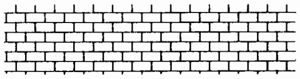
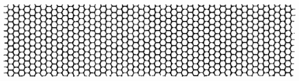
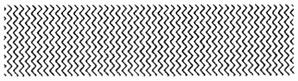
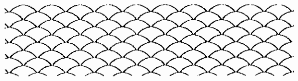
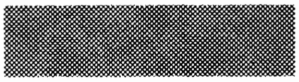
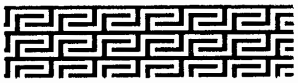
24 = Big checkerboard

25 = Little diamonds

26 = Big diamonds

27 = Little triangles



28=Big triangles	
29=Little bricks	
30=Big bricks	
31=Honeycomb	
32=Zigzag	
33=Fishscale	
34=Clovers	
35=Maze	

Colors

The IBM 3812 Printer supports two colors. Index 1 is black and index 0 is not displayed. These colors cannot be redefined.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available:

Fonts: 1=Courier.10.P0
2=Courier.bold.10.P0
3=Courier.italic.10.P0
4=Document.PSM.MLP
5=Document.PSM.P0

6=Boldface.PSM.MLP
7=Boldface.PSM.P0
8=Boldface.italic.PSM.MLP
9=Boldface.italic.PSM.P0
10=Document.PSM.P1
11=Document.PSM.P2
12=Boldface.PSM.P1
13=Boldface.PSM.P2
14=Boldface.italic.PSM.P1
15=Boldface.italic.PSM.P2
16=Essay.PSM.MLP
17=Essay.PSM.P0
18=Essay.PSM.P1
19=Essay.PSM.P2
20=Essay.bold.PSM.MLP
21=Essay.bold.PSM.P0
22=Essay.bold.PSM.P1
23=Essay.bold.PSM.P2
24=Essay.italic.PSM.MLP
25=Essay.italic.PSM.P0
26=Essay.italic.PSM.P1
27=Essay.italic.PSM.P2
28=Essay.light.PSM.MLP
29=Essay.light.PSM.P0
30=Essay.light.PSM.P1
31=Essay.light.PSM.P2
32=Sonoran-serif.8pt.MLP
33=Sonoran-serif.8pt.P0
34=Sonoran-serif.8pt.P1
35=Sonoran-serif.8pt.P2
36=Sonoran-serif.10pt.MLP
37=Sonoran-serif.10pt.P0
38=Sonoran-serif.10pt.P1
39=Sonoran-serif.10pt.P2
40=Sonoran-serif.12pt.MLP
41=Sonoran-serif.12pt.P0
42=Sonoran-serif.12pt.P1
43=Sonoran-serif.12pt.P2
44=Sonoran-serif.bold.10pt.MLP
45=Sonoran-serif.bold.10pt.P0
46=Sonoran-serif.bold.10pt.P1
47=Sonoran-serif.bold.10pt.P2

48=Sonoran-serif.bold.16pt.MLP
49=Sonoran-serif.bold.16pt.P0
50=Sonoran-serif.bold.16pt.P1
51=Sonoran-serif.bold.16pt.P2
52=Sonoran-serif.bold.18pt.MLP
53=Sonoran-serif.bold.18pt.P0
54=Sonoran-serif.bold.18pt.P1
55=Sonoran-serif.bold.18pt.P2
56=Sonoran-serif.bold.24pt.MLP
57=Sonoran-serif.bold.24pt.P0
58=Sonoran-serif.bold.24pt.P1
59=Sonoran-serif.bold.24pt.P2
60=Sonoran-serif.italic.10pt.MLP
61=Sonoran-serif.italic.10pt.P0
62=Sonoran-serif.italic.10pt.P1
63=Sonoran-serif.italic.10pt.P2
64=Courier.5.ASCII
65=Courier.5.P0
66=Courier.10.ASCII
67=Courier.10.MLP
68=Courier.10.P1
69=Courier.10.P2
70=Courier.12.MLP
71=Courier.12.P0
72=Courier.12.P1
73=Courier.12.P2
74=Courier.17.ASCII
75=Courier.17.P0
76=Courier.17ss.ASCII
77=Courier.17ss.P0
78=Courier.bold.5.ASCII
79=Courier.bold.5.P0
80=Courier.bold.10.ASCII
81=Courier.bold.17.ASCII
82=Courier.bold.17.P0
83=Courier.italic.10.MLP
84=Courier.italic.10.P1
85=Courier.italic.10.P2
86=Gothic-text.10.MLP
87=Gothic-text.10.P0
88=Gothic-text.10.P1
89=Gothic-text.12.MLP

```
 90=Gothic-text.12.P0
 91=Gothic-text.12.P1
 92=Gothic-text.13.MLP
 93=Gothic-text.13.P0
 94=Gothic-text.13.P1
 95=Gothic-text.15.MLP
 96=Gothic-text.15.P0
 97=Gothic-text.15.P1
 98=Gothic-text.20.MLP
 99=Gothic-text.20.P0
100=Gothic-text.20.P1
101=Gothic-text.27.MLP
102=Gothic-text.27.P0
103=Gothic-text.27.P1
104=Gothic-text.bold.10.MLP
105=Gothic-text.bold.10.P0
106=Gothic-text.bold.10.P1
107=Gothic-text.bold.12.MLP
108=Gothic-text.bold.12.P0
109=Gothic-text.bold.12.P1
110=Gothic-text.italic.12.MLP
111=Gothic-text.italic.12.P0
112=Gothic-text.italic.12.P1
113=Letter-gothic.12.MLP
114=Letter-gothic.12.P0
115=Letter-gothic.12.P1
116=Letter-gothic.12.P2
117=Letter-gothic.bold.12.MLP
118=Letter-gothic.bold.12.P0
119=Letter-gothic.bold.12.P1
120=Letter-gothic.bold.12.P2
121=Serif-text.10.MLP
122=Serif-text.10.P0
123=Serif-text.10.P1
124=Serif-text.12.MLP
125=Serif-text.12.P0
126=Serif-text.12.P1
127=Serif-text.15.MLP
128=Serif-text.15.P0
129=Serif-text.15.P1
130=Serif-text.bold.12.MLP
131=Serif-text.bold.12.P0
```

```
132=Serif-text.bold.12.P1
133=Serif-text.italic.10.MLP
134=Serif-text.italic.10.P0
135=Serif-text.italic.10.P1
136=Serif-text.italic.12.MLP
137=Serif-text.italic.12.P0
138=Serif-text.italic.12.P1
139=Prestige.10.MLP
140=Prestige.10.P0
141=Prestige.10.P1
142=Prestige.10.P2
143=Prestige.12.MLP
144=Prestige.12.P0
145=Prestige.12.P1
146=Prestige.12.P2
147=Prestige.bold.12.MLP
148=Prestige.bold.12.P0
149=Prestige.bold.12.P1
150=Prestige.bold.12.P2
151=Prestige.italic.12.MLP
152=Prestige.italic.12.P0
153=Prestige.italic.12.P1
154=Prestige.italic.12.P2
155=Orator.10.MLP
156=Orator.10.P0
157=Orator.10.P1
158=Orator.10.P2
159=Orator.bold.10.MLP
160=Orator.bold.10.P0
161=Orator.bold.10.P1
162=Orator.bold.10.P2
163=Roman-text.10.MLP
164=Roman-text.10.P0
165=Roman-text.10.P1
166=Script.12.MLP
167=Script.12.P0
168=Script.12.P1
169=Script.12.P2
170=APL.10.PC/APL
171=APL.20.PC/APL
172=OCR-A.10.P0
173=OCR-A.10.P2
174=OCR-B.10.P0
```

Sizes: Size is determined by font style.

Features: Underlining
Overstrike Mode
Superscript and Subscript

Device Specific Information

This driver supports a number of page options. Each option is selected via the system environment. In the following text, the commands to set these environmental parameters are shown for both the “sh” and “csh” shells. Refer to Appendix A, “Installing the Graphics Development Toolkit”, for more information about setting environmental parameters. The options are:

Page Orientation Option
This option can be set to either LANDSCAPE or PORTRAIT. The default for page orientation is PORTRAIT.

To set the page orientation option via the “sh” shell, enter:

```
ORIENTATION=LANDSCAPE;export ORIENTATION
```
(or)
```
ORIENTATION=PORTRAIT;export ORIENTATION
```

To set the page orientation option via the “csh” shell, enter:

```
setenv ORIENTATION LANDSCAPE
```
(or)
```
setenv ORIENTATION PORTRAIT
```

Page Size Option
This option can be set to either LETTER or LEGAL. The default for page size is LETTER. To set the page size option via the “sh” shell, enter:

```
PAPER=LETTER;export PAPER
```
(or)
```
PAPER=LEGAL;export PAPER
```

To set the page size option via the “csh” shell, enter:

```
setenv PAPER LETTER
```
(or)
```
setenv PAPER LEGAL
```

Tray Selection Option

This option can be set to either LOWER or UPPER. The default for tray selection is UPPER.

To set the tray selection option via the "sh" shell, enter:

```
TRAY=UPPER;export TRAY
```
(or)
```
TRAY=LOWER;export TRAY
```

To set the tray selection option via the "csh" shell, enter:

```
setenv TRAY UPPER
```
(or)
```
setenv TRAY LOWER
```

Number of Copies Option

This option can be set to any number between 1 and 999 (represented by "n" below).

To set the number of copies via the "sh" shell, enter:

```
COPIES=n;export COPIES
```

To set the number of copies via the "csh" shell, enter:

```
setenv COPIES n
```

Overlay Option

This option can be set to either TRUE or FALSE. When overlay mode is enabled, the picture generation command is suppressed. The default for overlay is FALSE.

To set the overlay option via the "sh" shell, enter:

```
OVERLAY=TRUE;export OVERLAY
```
(or)
```
OVERLAY=FALSE;export OVERLAY
```

To set the overlay option via the "csh" shell, enter:

```
setenv OVERLAY TRUE
```
(or)
```
setenv OVERLAY FALSE
```

Margins Option
This option can be set to either TRUE or FALSE. When margins are enabled, a 1/4 inch margin is given. The default for margins is FALSE.

To set the margins option via the "sh" shell, enter:

```
MARGIN=TRUE;export MARGIN
```
(or)
```
MARGIN=FALSE;export MARGIN
```

To set the margins option via the "csh" shell, enter:

```
setenv MARGIN TRUE
```
(or)
```
setenv MARGIN FALSE
```

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM 3812 Printer. If your application calls a routine that is not supported, an error will occur (error code –5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Update Workstation.

Cursor Control Routines

- Inquire Addressable Character Cells
- Output Cursor Addressable Text.

General Graphics Routines

- Inquire Color Representation
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text

- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Error Handling

- Inquire Error.

IBM 4201 Printer

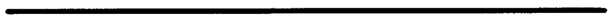
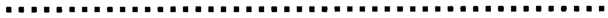
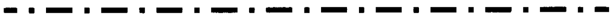
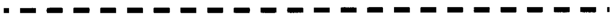
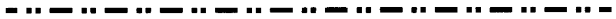
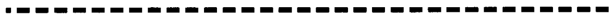
Filename: vdi4201

Features Supported

The following text describes each of the IBM 4201 Printer features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM 4201 Printer with one of seven line styles, selected with style indexes 1 thru 7:

1=Solid (round ends)	
2=Long Dash	
3=Dotted	
4=Dash Dotted	
5=Medium Dash	
6=Dash With Two Dots	
7=Short Dash	

Graphics Markers

The IBM 4201 Printer supports six Graphics Marker types, selected with type indexes 1 thru 6:

1=Dot	■
2=Cross	+
3=Star	✱
4=Square	□
5=X	×
6=Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following table shows the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios.

Graphics Marker Sizes

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	306	306
2	568	568
3	830	830
4	1091	1091
5	1353	1353

Graphics Text

The IBM 4201 Printer supports twelve sizes of Graphics Text. This text can be rotated on 0, 90, 180, and 270 degree baselines. The NDC units for each of the twelve text size indexes, for both Preserve and Non-Preserve Aspect ratios, are listed in the following tables:

Graphics Text Sizes
Preserve Aspect Ratio Mode

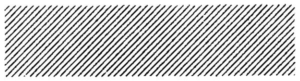
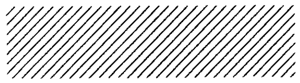
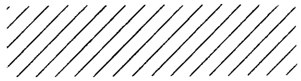
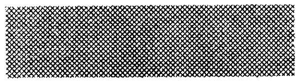
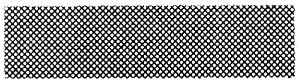
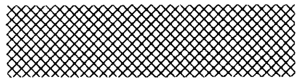
Index	Character Height	Character Width	Cell Height	Cell Width
1	306	262	350	315
2	611	524	699	629
3	917	786	1048	943
4	1222	1047	1397	1257
5	1528	1309	1746	1571
6	1833	1571	2095	1885
7	2138	1833	2444	2199
8	2444	2094	2793	2513
9	2749	2356	3142	2827
10	3055	2618	3491	3141
11	3360	2879	3840	3455
12	3666	3141	4189	3769

Graphics Text Sizes
Non-Preserve Aspect Ratio

Index	Character Height	Character Width	Cell Height	Cell Width
1	306	342	350	410
2	611	683	699	820
3	917	1024	1048	1229
4	1222	1366	1397	1639
5	1528	1707	1746	2048
6	1833	2048	2095	2458
7	2138	2390	2444	2868
8	2444	2731	2793	3277
9	2749	3072	3142	3687
10	3055	3414	3491	4096
11	3360	3755	3840	4506
12	3666	4096	4189	4916

Filled Areas

Filled areas, bars, pie slices and circles are all displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

1 = Right narrow diagonal	
2 = Right medium diagonal	
3 = Right wide diagonal	
4 = Narrow diagonal crosshatch	
5 = Medium diagonal crosshatch	
6 = Wide diagonal crosshatch	

Colors

The IBM 4201 Printer supports 2 colors: Index 1 is displayed in black ink and index 0 is not displayed. These colors cannot be redefined.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available:

Fonts:	1=Normal (default)
	2=Bold
Sizes:	1=17.6 characters per inch
	2=12 characters per inch
	3=10 characters per inch (default)
	4=8.3 characters per inch
	5=6 characters per inch
	6=5 characters per inch
Text Quality:	0-33=dot matrix quality (default)
	34-67=correspondence quality
	68-100=near letter quality
Features:	Underlining
	Overstrike Mode
	Pass Through Mode
	Superscript and Subscript
	Line Spacing
	Text Position

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM 4201 Printer. If your application calls a routine that is not supported, an error will occur (error code –5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Update Workstation.

Cursor Control Routines

- Inquire Addressable Character Cells
- Output Cursor Addressable Text.

General Graphics Routines

- Inquire Color Representation
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Error Handling

- Inquire Error.

IBM 5152 Graphics Printer

Filename: vdi5152

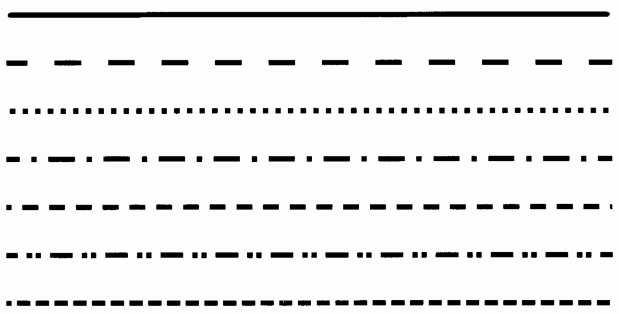
Features supported

The following text describes each of the IBM Graphics Printer features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM 5152 Graphics Printer with one of seven line styles, selected with style indexes 1 thru 7:

1 = Solid (round ends)
2 = Long Dash
3 = Dotted
4 = Dash Dotted
5 = Medium Dash
6 = Dash With Two Dots
7 = Short Dash



Graphics Markers

The IBM 5152 Graphics Printer supports six Graphics Marker types, selected with type indexes 1 thru 6:

1 = Dot	■
2 = Cross	+
3 = Star	✱
4 = Square	□
5 = X	✕
6 = Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following table shows the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios.

Graphics Marker Sizes		
Index	**Preserve Aspect Ratio**	**Non-Preserve Aspect Ratio**
1	306	306
2	568	568
3	830	830
4	1091	1091
5	1353	1353

Graphics Text

The IBM 5152 Graphics Printer supports 12 Graphics Text sizes. With this printer, Graphics Text can be rotated on 0, 90, 180, and 270 degree baselines. In the following tables, the 12 sizes are listed (in NDC units) for both Preserve and Non-Preserve Aspect Ratios.

Graphics Text Sizes
Preserve Aspect Ratio

Index	Character Height	Character Width	Cell Height	Cell Width
1	306	262	350	315
2	611	524	699	629
3	917	786	1048	943
4	1222	1047	1397	1257
5	1528	1309	1746	1571
6	1833	1571	2095	1885
7	2138	1833	2444	2199
8	2444	2094	2793	2513
9	2749	2356	3142	2827
10	3055	2618	3491	3141
11	3360	2879	3840	3455
12	3666	3141	4189	3769

Graphics Text Sizes
Non-Preserve Aspect Ratio

Index	Character Height	Character Width	Cell Height	Cell Width
1	306	342	350	410
2	611	683	699	820
3	917	1024	1048	1229
4	1222	1366	1397	1639
5	1528	1707	1746	2048
6	1833	2048	2095	2458
7	2138	2390	2444	2868
8	2444	2731	2793	3277
9	2749	3072	3142	3687
10	3055	3414	3491	4096
11	3360	3755	3840	4506
12	3666	4096	4189	4916

Filled Areas

Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

1 = Right narrow diagonal

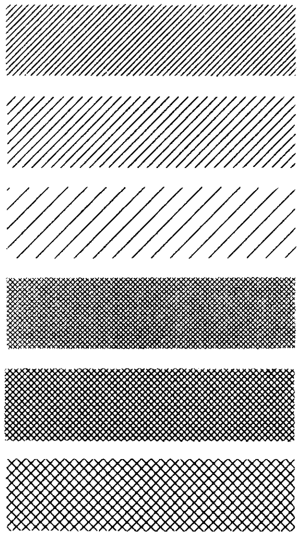
2 = Right medium diagonal

3 = Right wide diagonal

4 = Narrow diagonal crosshatch

5 = Medium diagonal crosshatch

6 = Wide diagonal crosshatch



Colors

The IBM 5152 Graphics Printer supports two colors. Number 1 is displayed with the black ribbon and number 0 is not displayed. These colors cannot be redefined.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available on the IBM 5152 Graphics Printer:

Fonts:	1=Normal (default) 2=Bold
Sizes:	1=17.16 characters per inch 2=10.00 characters per inch (default)
Text Quality:	(⟨50)=Bi-directional printing on (≥50)=Bi-directional printing off
Features:	Underlining Overstrike Mode Pass Through Mode Superscript and Subscript

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM 5152 Printer. If your application calls a routine that is not supported, an error will occur (error code -5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Update Workstation.

Cursor Control Routines

- Inquire Addressable Character Cells
- Output Cursor Addressable Text.

General Graphics Routines

- Inquire Color Representation
- Set Color Representation
- Set Writing Mode (supports all 16 modes).

Graphics Primitives

- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size

- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Error Handling

- Inquire Error.

IBM 5182 Color Printer

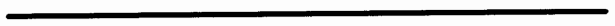
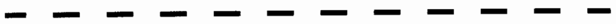
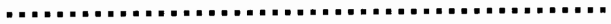
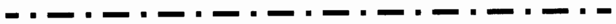
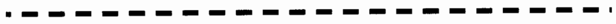
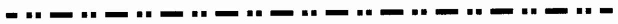
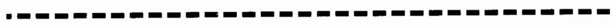
Filename: vdi5182

Features Supported

The following text describes each of the IBM 5182 Color Printer features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM 5182 Color Printer with one of seven line styles, selected with style indexes 1 thru 7:

1=Solid (round ends)	
2=Long Dash	
3=Dotted	
4=Dash Dotted	
5=Medium Dash	
6=Dash With Two Dots	
7=Short Dash	

Graphics Markers

The IBM 5182 Color Printer supports six Graphics Marker types, selected with type indexes 1 thru 6:

1=Dot	■
2=Cross	+
3=Star	✱
4=Square	□
5=X	✕
6=Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following table shows the NDC units for the five Graphics Marker size indexes, in both Preserve and Non-Preserve Aspect Ratios.

Graphics Marker Sizes

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	268	268
2	497	497
3	726	726
4	955	955
5	1184	1184

Graphics Text

The IBM 5182 Color Printer supports 12 Graphics Text sizes. With this printer, Graphics Text can be rotated on 0, 90, 180, and 270 degree baselines. The 12 sizes are listed, in NDC units, for both Preserve Aspect Ratio and Non-Preserve aspect ratio in the following tables.

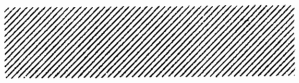
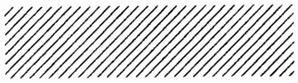
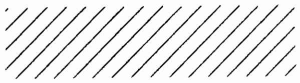
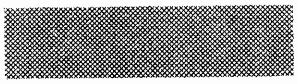
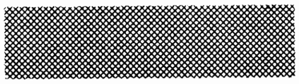
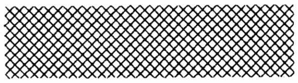
Graphics Text Sizes
Preserve Aspect Ratio

Index	Character Height	Character Width	Cell Height	Cell Width
1	268	191	306	230
2	535	382	612	459
3	803	573	917	688
4	1070	764	1223	917
5	1337	955	1528	1146
6	1605	1146	1834	1375
7	1872	1337	2139	1605
8	2139	1528	2445	1834
9	2407	1719	2750	2063
10	2674	1910	3056	2292
11	2941	2101	3361	2521
12	3209	2292	3667	2750

Graphics Text Sizes Non-Preserve Aspect Ratio				
Index	Character Height	Character Width	Cell Height	Cell Width
1	268	250	306	300
2	535	500	612	600
3	803	750	917	900
4	1070	1000	1223	1199
5	1337	1249	1528	1499
6	1605	1499	1834	1799
7	1872	1749	2139	2098
8	2139	1999	2445	2398
9	2407	2248	2750	2698
10	2674	2498	3056	2998
11	2941	2748	3361	3297
12	3209	2998	3667	3597

Filled Areas

Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

1=Right narrow diagonal	
2=Right medium diagonal	
3=Right wide diagonal	
4=Narrow diagonal crosshatch	
5=Medium diagonal crosshatch	
6=Wide diagonal crosshatch	

Colors

The IBM 5182 Color Printer supports eight colors. These colors cannot be redefined. Each color is associated with a color index, as follows:

0=Background
1=Black
2=Red
3=Green
4=Blue
5=Yellow
6=Orange
7=Violet

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following alpha text capabilities are available on the IBM 5182 Color Printer:

Fonts:	1=Normal (default) 2=Bold
Sizes:	1=17.16 characters per inch 2=12 character per inch 3=10 characters per inch (default) 4=5 characters per inch
Text Quality:	(⟨50)=Bi-directional printing on (≥50)=Bi-directional printing off
Features:	Underlining Overstrike Mode Pass Through Mode Superscript and Subscript

Device Specific Information

The IBM 5182 Color Printer driver allows the user to change the paper width and ribbon selection independently of each other. The valid options for paper are WIDE (13 inch output) and NARROW (8 inch output). The default is Narrow paper. The ribbon has three options: BLACK (color number 0 and 1 are valid), RGB (color numbers 0 to 4 are valid), and the PROCESS ribbon (color numbers 0 to 7 are valid). The default ribbon is the PROCESS ribbon.

Note: To use the WIDE paper option, the appropriate printer DIP switch must be set properly. For more information on setting printer DIP switches, refer to your printer manual.

Some device options can be selected by exporting them to the operating system environment.

To select the paper size, with the “sh” shell, enter:

```
PAPER=WIDE;export PAPER
```
(or)
```
PAPER=NARROW;export PAPER
```

To select the paper size, with the “csh” shell, enter:

```
setenv PAPER WIDE
```
(or)
```
setenv PAPER NARROW
```

To select the ribbon, with the “sh” shell, enter:

```
RIBBON=BLACK;export RIBBON
```
(or)
```
RIBBON=RGB;export RIBBON
```
(or)
```
RIBBON=PROCESS;export RIBBON
```

To select the ribbon, with the “csh” shell, enter:

```
setenv RIBBON BLACK
```
(or)
```
setenv RIBBON RGB
```
(or)
```
setenv RIBBON PROCESS
```

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM 5182 Printer. If your application calls a routine that is not supported, an error will occur (error code -5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Update Workstation.

Cursor Control Routines

- Inquire Addressable Character Cells
- Output Cursor Addressable Text.

General Graphics Routines

- Inquire Color Representation
- Set Color Representation
- Set Writing Mode (supports all 16 modes).

Graphics Primitives

- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Error Handling

- Inquire Error.

IBM 6180 Color Plotter

Filename: vdi6180

Features Supported

The following text describes each of the IBM 6180 Color Plotter features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM 6180 Color Plotter with one of seven line styles, selected with style indexes 1 thru 7:

1 = Solid (round ends)

2 = Long Dash

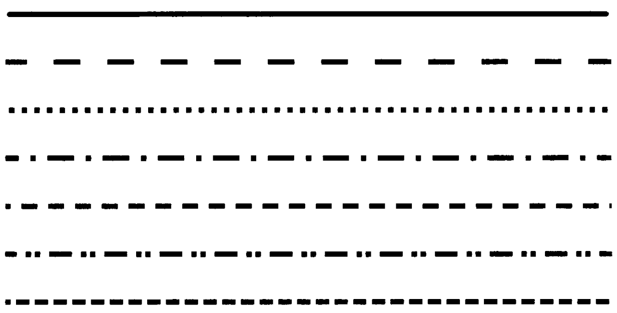
3 = Dotted

4 = Dash Dotted

5 = Medium Dash

6 = Dash With Two Dots

7 = Short Dash



Graphics Markers

The IBM 6180 Color Plotter supports six Graphics Marker types, selected with type indexes 1 thru 6:

1=Dot	■
2=Cross	+
3=Star	✱
4=Square	□
5=X	×
6=Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following tables show the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios and in two sizes of paper.

Graphics Marker Sizes
"A" Size Paper

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	195	262
2	385	519
3	576	776
4	767	1033
5	958	1290

Graphics Marker Sizes "A4" Size Paper		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	184	262
2	364	519
3	545	776
4	725	1033
5	905	1290

Graphics Text

The IBM 6180 Color Plotter supports continuous character scaling. Graphics Text can be rotated from 0 to 359.9 degrees, in increments of one tenth of a degree. The following character sets are available as Graphics Text fonts on the IBM 6180 Color Plotter:

1=ANSI ASCII
2=9825 Character Set
3=French/German
4=Scandinavian
5=Spanish/Latin American

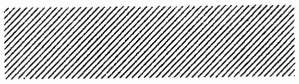
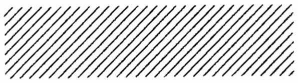
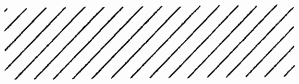
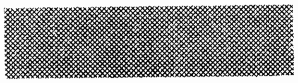
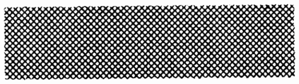
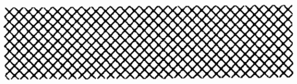
If the Graphics Enhancement Cartridge is installed the following Graphics Text fonts will also be available:

6=JIS ASCII
7=Roman (8 Extensions)
8=Katakana
9=International Version
10=Swedish
11=Swedish for names
12=Norway (Version I)
13=German

14=French
15=United Kingdom
16=Italian
17=Spanish
18=Portuguese
19=Norway (Version II)

Filled Areas

Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

1=Right narrow diagonal	
2=Right medium diagonal	
3=Right wide diagonal	
4=Narrow diagonal crosshatch	
5=Medium diagonal crosshatch	
6=Wide diagonal crosshatch	

Colors

Color indexes 1 through 8 are mapped to pen stations 1 through 8 of the IBM 6180 Color Plotter. Color 0 is not displayed.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available on the IBM 6180 Color Plotter:

Fonts:	(See Alpha Text Font Index Table)
Sizes:	Continuous character scaling, default is 66 characters down page
Features:	Underlining Overstrike Mode Superscript and Subscript Color

Alpha Text Font Index

Description	Normal	Bold	Italics
ANSI ASCII	1	2	3
Reserved for VDI	4	5	6
9825 Character Set	7	8	9
French/German	10	11	12
Scandinavian	13	14	15
Spanish/Latin American	16	17	18

If the Graphics Enhancement Cartridge is installed, the following fonts are also available:

Alpha Text Font Index			
Description	Normal	Bold	Italics
JIS ASCII	19	20	21
Roman - 8 Extensions	22	23	24
Katakana	25	26	27
International Version	28	29	30
Swedish	31	32	33
Swedish for names	34	35	36
Norway (Version I)	37	38	39
German	40	41	42
French	43	44	45
United Kingdom	46	47	48
Italian	49	50	51
Spanish	52	53	54
Portuguese	55	56	57
Norway (Version II)	58	59	60

Request Locator

The pen holder location is used to indicate the plotter pen's coordinates to VDI. The pen holder is moved by pressing the position keys on the plotter's front panel. When the pen holder is at the desired location, the point can be selected by pressing the plotter's **Enter** button. This causes the coordinates of the point to be transmitted back to the user program.

Device Specific Information

This driver will drive the plotter in both A and A4 size modes. The paper size can be selected by setting the rear DIP switch. This should be done before running your application.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM 6180 Plotter. If your application calls a routine that is not supported, an error will occur (error code –5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Set Pen Speed
- Update Workstation.

Cursor Control Routines

- Inquire Addressable Character Cells
- Output Cursor Addressable Text.

General Graphics Routines

- Inquire Color Representation
- Set Color Representation
- Set Writing Mode (overstrike mode only).

Graphics Primitives

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index

- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Locator (request mode).

Error Handling

- Inquire Error.

IBM 7371 Color Plotter

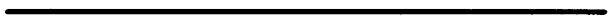
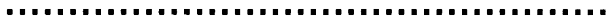
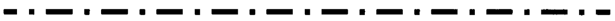
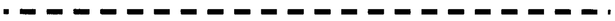
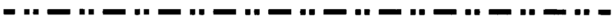
Filename: vdi7371

Features Supported

The following text describes each of the IBM 7371 Color Plotter features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM 7371 Color Plotter with one of six line styles, selected with style indexes 1 thru 6:

1=Solid (round ends)	
2=Long Dash	
3=Dotted	
4=Dash Dotted	
5=Medium Dash	
6=Dash With Two Dots	

Graphics Markers

The IBM 7371 Color Plotter supports six Graphics Marker types, selected with type indexes 1 thru 6:

1=Dot	■
2=Cross	+
3=Star	✱
4=Square	□
5=X	✕
6=Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following table shows the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios.

Graphics Marker Sizes

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	195	262
2	385	519
3	576	776
4	767	1033
5	958	1290

Graphics Text

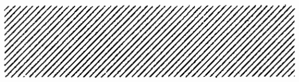
The IBM 7371 Color Plotter supports continuous character scaling. Graphics Text can be rotated from 0 to 359.9 degrees, in increments of one tenth of a degree. There are five character fonts available for Graphics Text on the IBM 7371 Color Plotter, as follows:

1=ANSI ASCII
2=9825 Character Set
3=French/German
4=Scandinavian
5=Spanish/Latin American

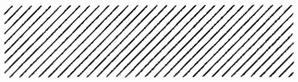
Filled Areas

Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

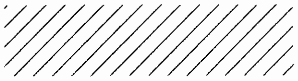
1=Right narrow diagonal



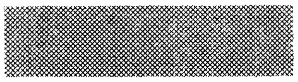
2=Right medium diagonal



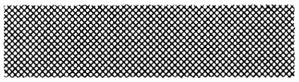
3=Right wide diagonal



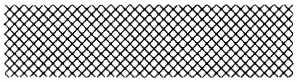
4=Narrow diagonal crosshatch



5=Medium diagonal crosshatch



6=Wide diagonal crosshatch



Colors

Color 1 always is located in pen station 1. It is assumed to be a black pen. By default, color index 2 is located in pen station 2. If the user program requests a color index other than what is currently in pen stations 1 and 2, the user is sent a prompt to insert the requested pen into pen station 2. Color 0 is not displayed.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available on the IBM 7371 Color Plotter:

Fonts:	(See Alpha Text Font Index Table)
Sizes:	Continuous character scaling, default is 66 characters down page
Features:	Underlining Overstrike Mode Superscript and Subscript Color

Alpha Text Font Index			
Description	Normal	Bold	Italics
ANSI ASCII	1	2	3
Reserved for VDI	4	5	6
9825 Character Set	7	8	9
French/German	10	11	12
Scandinavian	13	14	15
Spanish/Latin American	16	17	18

Request Locator

The pen holder location is used to indicate the plotter pen's coordinates to VDI. The pen holder is moved by pressing the position keys on the plotter's front panel. When the pen holder is at the desired location, the point can be selected by pressing the plotter's **Enter** button. This causes the coordinates of the point to be transmitted back to the user program.

Device Specific Information

This driver drives the plotter in small paper (8½ × 11) mode only.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM 7371 Plotter. If your application calls a routine that is not supported, an error will occur (error code -5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Set Pen Speed
- Update Workstation.

Cursor Control Routines

- Inquire Addressable Character Cells
- Output Cursor Addressable Text.

General Graphics Routines

- Inquire Color Representation
- Set Color Representation
- Set Writing Mode (overstrike mode only).

Graphics Primitives

- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes

- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position

- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Locator (request mode).

Error Handling

- Inquire Error.

IBM 7372 Color Plotter

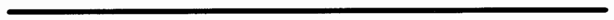
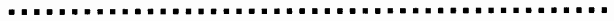
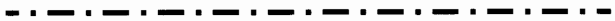
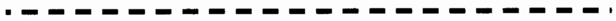
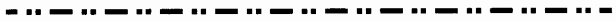
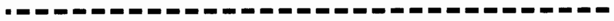
Filename: vdi7372

Features Supported

The following text describes each of the IBM 7372 Color Plotter features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM 7372 Color Plotter with one of seven line styles, selected with style indexes 1 thru 7:

1=Solid (round ends)	
2=Long Dash	
3=Dotted	
4=Dash Dotted	
5=Medium Dash	
6=Dash With Two Dots	
7=Short Dash	

Graphics Markers

The IBM 7372 Color Plotter supports six Graphics Marker types, selected with type indexes 1 thru 6:

1=Dot	■
2=Cross	+
3=Star	✱
4=Square	□
5=X	×
6=Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following tables show the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios and in two sizes of paper.

Graphics Marker Sizes
"A" Size Paper

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	121	193
2	239	383
3	357	573
4	475	762
5	593	952

Graphics Marker Sizes
"B" Size Paper

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	193	252
2	383	498
3	573	745
4	762	992
5	952	1239

Graphics Text

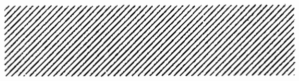
The IBM 7372 Color Plotter supports continuous character scaling. Graphics Text can be rotated from 0 to 359.9 degrees, in increments of one tenth of a degree. There are 19 character sets available as Graphics Text fonts on the IBM 7372 Color Plotter, as follows:

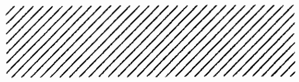
1=ANSI ASCII
2=9825 Character Set
3=French/German
4=Scandinavian
5=Spanish/Latin American

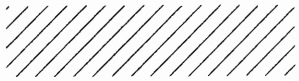
6=JIS ASCII
7=Roman (8 Extensions)
8=Katakana
9=International Version
10=Swedish
11=Swedish for names
12=Norway (Version I)
13=German
14=French
15=United Kingdom
16=Italian
17=Spanish
18=Portuguese
19=Norway (Version II)

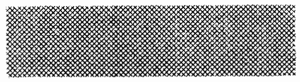
Filled Areas

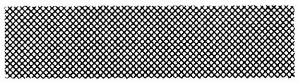
Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

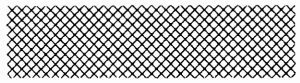
1=Right narrow diagonal 

2=Right medium diagonal 

3=Right wide diagonal 

4=Narrow diagonal crosshatch 

5=Medium diagonal crosshatch 

6=Wide diagonal crosshatch 

Colors

Color indexes 1 through 6 are mapped to pen stations 1 through 6 of the IBM 7372 Color Plotter. Color 0 is not displayed.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available on the IBM 7372 Color Plotter:

Fonts:	(See Alpha Text Font Index Table)
Sizes:	Continuous character scaling, default is 66 characters down page
Features:	Underlining Overstrike Mode Superscript and Subscript Color

Alpha Text Font Index

Description	Normal	Bold	Italics
ANSI ASCII	1	2	3
Reserved for VDI	4	5	6
9825 Character Set	7	8	9
French/German	10	11	12
Scandinavian	13	14	15
Spanish/Latin American	16	17	18
JIS ASCII	19	20	21
Roman - 8 Extensions	22	23	24
Katakana	25	26	27
International Version	28	29	30

Alpha Text Font Index			
Description	Normal	Bold	Italics
Swedish	31	32	33
Swedish for names	34	35	36
Norway (Version I)	37	38	39
German	40	41	42
French	43	44	45
United Kingdom	46	47	48
Italian	49	50	51
Spanish	52	53	54
Portuguese	55	56	57
Norway (Version II)	58	59	60

Request Locator

The pen holder location is used to indicate the plotter pen's coordinates to VDI. The pen holder is moved by pressing the position keys on the plotter's front panel. When the pen holder is at the desired location, the point can be selected by pressing the plotter's **Enter** button. This causes the coordinates of the point to be transmitted back to the user program.

Device Specific Information

This driver will drive the plotter in both A and B size modes. The paper size can be selected by setting the rear DIP switch or by pressing the plotter's **SIZE** button simultaneously with the plotter's **Enter** button. This should be done before running your application.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM 7372 Plotter. If your application calls a routine that is not supported, an error will occur (error code -5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Set Pen Speed
- Update Workstation.

Cursor Control Routines

- Inquire Addressable Character Cells
- Output Cursor Addressable Text.

General Graphics Routines

- Inquire Color Representation
- Set Color Representation
- Set Writing Mode (overstrike mode only).

Graphics Primitives

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style

- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Locator (request mode).

Error Handling

- Inquire Error.

IBM 7374, 7375-1, 7375-2 Plotters

Filename: vdi7375

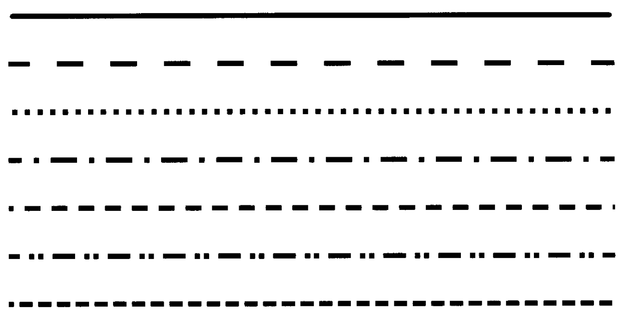
Features Supported

The following text describes each of the IBM 7374, 7375-1, or 7375-2 Plotter features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM 7374, 7375-1, and 7375-2 Plotters with one of seven line styles, selected with style indexes 1 thru 7:

1 = Solid (round ends)
2 = Long Dash
3 = Dotted
4 = Dash Dotted
5 = Medium Dash
6 = Dash With Two Dots
7 = Short Dash



Graphics Markers

The IBM 7374, 7375-1, and 7375-2 Plotters support six Graphics Marker types, selected with type indexes 1 thru 6:

1 = Dot	■
2 = Cross	+
3 = Star	✱
4 = Square	□
5 = X	✕
6 = Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following tables show the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios, for each of the available paper size options.

Graphics Markers
"A" Size Paper

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	177	177
2	351	351
3	524	524
4	697	697
5	870	870

Graphics Markers
"B" Size Paper

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	315	481
2	627	957
3	938	1433
4	1250	1910
5	1562	2386

Graphics Markers "C" Size Paper		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	217	217
2	433	433
3	648	648
4	863	863
5	1079	1079

Graphics Markers "D1" Size Paper		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	323	501
2	644	1000
3	964	1500
4	1285	1998
5	1606	2498

Graphics Markers "D2" Size Paper		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	190	190
2	379	379
3	568	568
4	757	757
5	946	946

Graphics Markers “E” Size Paper		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	325	416
2	649	829
3	973	1243
4	1297	1657
5	1621	2071

Dimensions of Paper Sizes			
Paper Size	Paper Width	Paper Length	Orientation
A	8.5″	11″	(length along platen)
B	11″	17″	(width along platen)
C	17″	22″	(length along platen)
D1	22″	34″	(width along platen)
D2	22″	34″	(length along platen)
E	34″	44″	(width along platen)

Note: For roll paper, use sheet paper values of the paper size with the same width as the roll.

Graphics Text

These plotters support continuous character scaling. Graphics Text can be rotated from 0 to 359.9 degrees, in increments of one tenth of a degree. There are 19 character sets available as Graphics Text fonts on these plotters, as follows:

1=ANSI ASCII
2=9825 Character set
3=French/German
4=Scandanavian
5=Spanish/Latin American
6=Special Symbols
7=JIS ASCII
8=Roman, 8 Extensions
9=Katakana
10=ISO IRV
11=ISO Swedish
12=ISO Swedish (names)
13=ISO Norway V1
14=ISO German
15=ISO French
16=ISO United Kingdom
17=ISO Italian
18=ISO Spanish
19=ISO Portuguese

Filled Areas

Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

1=Right narrow diagonal	▨
2=Right medium diagonal	▨
3=Right wide diagonal	▨
4=Narrow diagonal crosshatch	▩
5=Medium diagonal crosshatch	▩
6=Wide diagonal crosshatch	▩

Colors

Color indices 1 through 8 are mapped to pen stations 1 through 8 of the plotter. Color 0 is not displayed.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available on the plotter:

Fonts:	(See Alpha Text Font Index Table)
Sizes:	One size can be selected, default is 66 characters down page

Features:	Underlining
	Overstrike Mode
	Superscript and Subscript
	Color

Alpha Text Font Index

Font	Normal	Bold	Italics
ANSI ASCII	1	2	3
Reserved for VDI	4	5	6
9825 Character Set	7	8	9
French/German	10	11	12
Scandanavian	13	14	15
Spanish/Latin American	16	17	18
Special Symbols	19	20	21
JIS ASCII	22	23	24
Roman, 8 Extensions	25	26	27
Katakana	28	29	30
ISO IRV	31	32	33
ISO Swedish	34	35	36
ISO Swedish (for names)	37	38	39
ISO Norway V1	40	41	42
ISO German	43	44	45
ISO French	46	47	48
ISO United Kingdom	49	50	51
ISO Italian	52	53	54
ISO Spanish	55	56	57
ISO Portuguese	58	59	60

Request Locator

When request locator is invoked, the LED on the **Enter** button is lit. The joystick is used to indicate the point to be input. When the pen holder is at the desired location, the point can be selected by pressing the **Enter** button. This causes the coordinates of the point to be transmitted to the user program.

Device Specific Information

Only one paper size may be used during any individual workstation session. The workstation must be closed and reopened when changing the paper size.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM 7374 and 7375 Plotters. If your application calls a routine that is not supported, an error will occur (error code -5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Set Pen Speed
- Update Workstation.

Cursor Control Routines

- Inquire Addressable Character Cells
- Output Cursor Addressable Text.

General Graphics Routines

- Inquire Color Representation
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes

- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position

- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Locator (request mode).

Error Handling

- Inquire Error.

IBM Advanced Monochrome Graphics Display

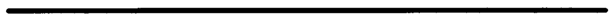
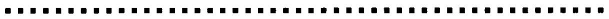
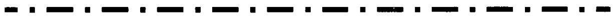
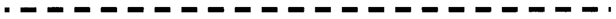
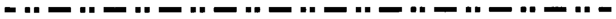
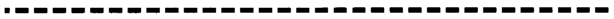
Filename: vdiamg

Features Supported

The following text describes each of the IBM Advanced Monochrome Graphics Display features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM Advanced Monochrome Graphics Display with one of seven line styles, selected with style indexes 1 thru 7:

1=Solid (round ends)	
2=Long Dash	
3=Dotted	
4=Dash Dotted	
5=Medium Dash	
6=Dash With Two Dots	
7=Short Dash	

Graphics Markers

The IBM Advanced Monochrome Graphics Display supports six Graphics Marker types, selected with type indexes 1 thru 6:

1=Dot	■
2=Cross	+
3=Star	✱
4=Square	□
5=X	✕
6=Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following table shows the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios.

Graphics Marker Sizes

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	319	448
2	592	832
3	865	1216
4	1138	1600
5	1411	1984

Graphics Text

Graphics Text fonts are selected by index number. Index number 1 is reserved for the default font. All other index numbers are derived from an alphabetical list of font files that end with the .9X20 extension and reside in **/etc/vtm.** For example, the first file in the alphabetical list would be index number 2, the second would be index number 3 and so on.

Graphics Text can be rotated on 0, 90, 180, and 270 degree baselines. The IBM Advanced Monochrome Graphics Display supports five sizes in each different display mode. These sizes are listed below in NDC units for each of the modes:

Graphics Text Sizes
Preserve Aspect Ratio

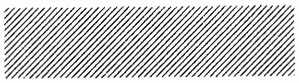
Index	Character Height	Character Width	Cell Height	Cell Width
1	729	365	911	410
2	1457	729	1821	820
3	2185	1093	2731	1229
4	2913	1457	3641	1639
5	3641	1821	4552	2048

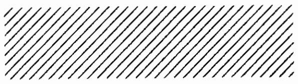
Graphics Text Sizes
Non-Preserve Aspect Ratio

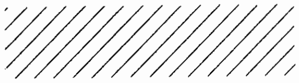
Index	Character Height	Character Width	Cell Height	Cell Width
1	1024	365	1280	410
2	2048	729	2560	820
3	3072	1093	3840	1229
4	4096	1457	5120	1639
5	5120	1821	6400	2048

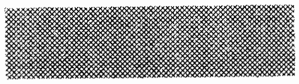
Filled Areas

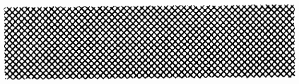
Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

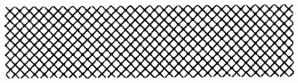
1=Right narrow diagonal 

2=Right medium diagonal 

3=Right wide diagonal 

4=Narrow diagonal crosshatch 

5=Medium diagonal crosshatch 

6=Wide diagonal crosshatch 

Colors

In graphics mode, there are two available colors that can be used to display graphics primitives. Color index 0 is displayed as black, and color index 1 is displayed as white. These colors cannot be redefined.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available:

Fonts:	Font index 1 is the standard device font, fonts 2 through 6 are reserved for VDI, and font indexes 7 and upwards are taken from an alphabetical list of files that end with the .9X20 extension and reside in **/etc/vtm**.
Sizes:	One Size
Features:	Underlining Overstrike Mode Superscript and Subscript Line Spacing

Request Locator

When locator is invoked, a tracking cross appears on the display at the initial locator position. The cross can be moved with the IBM RT PC Mouse or by pressing one of the four arrow keys on the keyboard.

Initially, the cross moves in large increments. Pressing the **Insert** key toggles the distance between large movements and small movements. When the cross is at the desired location, the point can be selected by pressing any alpha key on the keyboard. This causes the coordinates of the point to be transmitted back to the user program. If desired, the device will perform an inking function. When the locator is terminated, a line from the initial position to the desired position is drawn honoring the current line attributes such as color and line style.

Also the device performs rubberbanding if desired. There are two types of rubberbanding supported, lines and boxes. If rubberbanding lines are desired, then a line will be drawn from the initial locator position to the current position of the graphics cursor. The line changes dynamically as the cursor is moved. When the locator is terminated, the line is removed. If rubberband rectangle is specified, a rectangle is displayed with one corner at the initial locator position and the opposite at the current position of the graphics cursor. The rectangle changes dynamically as the cursor is moved. When the locator is terminated, the rectangle is removed from the display.

Request Choice

The function keys **F1** thru **F12** on the keyboard, in conjunction with the **SHIFT, CTRL,** and **ALT** keys, are used to enter choice input. Thus, a total of 48 different inputs can be generated. Pressing keys **F1** thru **F12** alone will generate inputs 1 thru 12, respectively. If the **SHIFT** key is held down, while pressing keys **F1** thru **F12,** inputs 13 thru 24 will be generated. If the **CTRL** key is held down, and the twelve keys are pressed, inputs 25 thru 36 will be generated. If the **ALT** key is held down, and the twelve keys are pressed, inputs 37 thru 48 will be generated.

Request String

The keyboard is used to enter strings. The string is terminated by the **Enter** key.

Cursor Addressable Text

Cursor addressable text is supported. The device must be in Cursor Addressing Mode before it can perform any cursor control functions. To display graphics primitives, the device must be removed from Cursor Addressing Mode. The Reverse Video attribute is supported.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM Advanced Monochrome Graphics Display. If your application calls a routine that is not supported, an error will occur (error code -5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Update Workstation.

Paging Routines

- Copy Page
- Inquire Page
- Set Page.

Pel Routines

- Copy Pels
- Get Pels
- Put Pels.

Cursor Control Routines

- Cursor Down
- Cursor Home
- Cursor Left
- Cursor Right
- Cursor Up
- Direct Cursor Address
- Enter Cursor Addressing Mode
- Erase to End of Line
- Erase to End of Screen
- Exit Cursor Addressing Mode
- Inquire Addressable Character Cells
- Inquire Current Cursor Text Address
- Output Cursor Addressable Text
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index.

General Graphics Routines

- Display Graphic Input Cursor
- Inquire Color Representation
- Remove Graphic Input Cursor
- Set Background Color Index
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Choice (request mode)
- Input Locator (request mode)
- Input String (request mode)
- Input String (sample mode)
- Read Cursor Movement Keys
- Set Line Edit Characters.

Error Handling

- Inquire Error.

IBM Advanced Color Graphics Display

Filename: vdiacg

Features Supported

The following text describes each of the IBM Advanced Color Graphics Display features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM Advanced Color Graphics Display with one of seven line styles, selected with style indexes 1 thru 7:

1 = Solid (round ends)	──────────────
2 = Long Dash	— — — — — — — — — — — —
3 = Dotted	....
4 = Dash Dotted	- · — · — · — · — · — · — · — · — · — · -
5 = Medium Dash	- -
6 = Dash With Two Dots	- ·· — ·· — ·· — ·· — ·· — ·· — ·· — ·· — ·· -
7 = Short Dash	------------------------------------

Graphics Markers

The IBM Advanced Color Graphics Display supports six Graphics Marker types, selected with type indexes 1 thru 6:

1 = Dot	■
2 = Cross	+
3 = Star	✱
4 = Square	□
5 = X	×
6 = Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following table shows the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios.

Graphics Marker Sizes

Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	319	448
2	592	832
3	865	1216
4	1138	1600
5	1411	1984

Graphics Text

Graphics Text fonts are selected by index number. Index number 1 is reserved for the default font. All other index numbers are derived from an alphabetical list of font files that end with the .9X20 extension and reside in **/etc/vtm.** For example, the first file in the alphabetical list would be index number 2, the second would be index number 3 and so on.

Graphics Text can be rotated on 0, 90, 180, and 270 degree baselines. The IBM Advanced Color Graphics Display supports five sizes in each different display mode. These sizes are listed below in NDC units for each of the modes:

Graphics Text Sizes
Preserve Aspect Ratio

Index	Character Height	Character Width	Cell Height	Cell Width
1	729	365	911	410
2	1457	729	1821	820
3	2185	1093	2731	1229
4	2913	1457	3641	1639
5	3641	1821	4552	2048

Graphics Text Sizes Non-Preserve Aspect Ratio				
Index	Character Height	Character Width	Cell Height	Cell Width
1	1024	365	1280	410
2	2048	729	2560	820
3	3072	1093	3840	1229
4	4096	1457	5120	1639
5	5120	1821	6400	2048

Filled Areas

Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

1=Right narrow diagonal

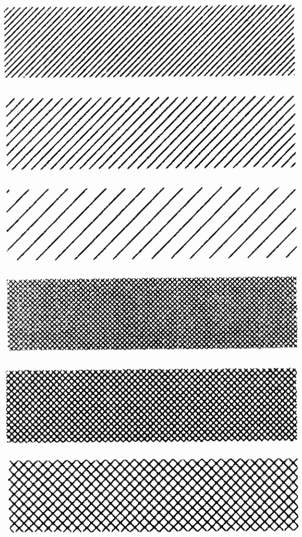
2=Right medium diagonal

3=Right wide diagonal

4=Narrow diagonal crosshatch

5=Medium diagonal crosshatch

6=Wide diagonal crosshatch



Colors

In graphics mode, there are 16 available colors that can be used to display graphics primitives. They are defined as follows:

0	Black
1	White
2	Red
3	Green
4	Blue
5	Yellow
6	Cyan
7	Magenta
8-15	White

These indices can be redefined into a color chosen from a palette of 64 available colors, using the Set Color Representation function. Each of the Red, Green, and Blue components can have values of 0, 250, 500, 750, or 1000.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available:

Fonts:	Font index 1 is the standard device font, fonts 2 through 6 are reserved for VDI, and font indexes 7 and upwards are taken from an alphabetical list of files that end with the .9X20 extension and reside in **/etc/vtm.**
Sizes:	One Size
Features:	Underlining Overstrike Mode Superscript and Subscript Line Spacing

Request Locator

When locator is invoked, a tracking cross appears on the display at the initial locator position. The cross can be moved with the IBM RT PC Mouse or by pressing one of the four arrow keys on the keyboard.

Initially, the cross moves in large increments. Pressing the **Insert** key toggles the distance between large movements and small movements. When the cross is at the desired location, the point can be selected by pressing any alpha key on the keyboard. This causes the coordinates of the point to be transmitted back to the user program.

If desired, the device will perform an inking function. When the locator is terminated, a line from the initial position to the desired position is drawn honoring the current line attributes such as color and line style.

Also the device performs rubberbanding if desired. There are two types of rubberbanding supported, lines and boxes. If rubberbanding lines are desired, then a line will be drawn from the initial locator position to the current position of the graphics cursor. The line changes dynamically as the cursor is moved. When the locator is terminated, the line is removed. If rubberband rectangle is specified, a rectangle is displayed with one corner at the initial locator position and the opposite at the current position of the graphics cursor. The rectangle changes dynamically as the cursor is moved. When the locator is terminated, the rectangle is removed from the display.

Request Choice

The function keys **F1** thru **F12** on the keyboard, in conjunction with the **SHIFT, CTRL,** and **ALT** keys, are used to enter choice input. Thus, a total of 48 different inputs can be generated. Pressing keys **F1** thru **F12** alone will generate inputs 1 thru 12, respectively. If the **SHIFT** key is held down, while pressing keys **F1** thru **F12,** inputs 13 thru 24 will be generated. If the **CTRL** key is held down, and the twelve keys are pressed, inputs 25 thru 36 will be generated. If the **ALT** key is held down, and the twelve keys are pressed, inputs 37 thru 48 will be generated.

Request String

The keyboard is used to enter strings. The string is terminated by the **Enter** key.

Cursor Addressable Text

Cursor addressable text is supported. The device must be in Cursor Addressing Mode before it can perform any cursor control functions. To display graphics primitives, the device must be removed from Cursor Addressing Mode. The Reverse Video attribute is supported.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM Advanced Color Graphics Display. If your application calls a routine that is not supported, an error will occur (error code –5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Update Workstation.

Paging Routines

- Copy Page
- Inquire Page
- Set Page.

Pel Routines

- Copy Pels
- Get Pels
- Put Pels.

Cursor Control Routines

- Cursor Down
- Cursor Home
- Cursor Left
- Cursor Right
- Cursor Up
- Direct Cursor Address
- Enter Cursor Addressing Mode
- Erase to End of Line
- Erase to End of Screen

- Exit Cursor Addressing Mode
- Inquire Addressable Character Cells
- Inquire Current Cursor Text Address
- Output Cursor Addressable Text
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index.

General Graphics Routines

- Display Graphic Input Cursor
- Inquire Color Representation
- Remove Graphic Input Cursor
- Set Background Color Index
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Choice (request mode)
- Input Locator (request mode)
- Input String (request mode)
- Input String (sample mode)
- Read Cursor Movement Keys
- Set Line Edit Characters.

Error Handling

- Inquire Error.

IBM Extended Monochrome Graphics Display

Filename: vdiemg

Features Supported

The following text describes each of the IBM Extended Monochrome Graphics Display features that are supported by this device driver.

Polylines

Lines and arcs can be drawn on the IBM Extended Monochrome Graphics Display with one of seven line styles, selected with style indexes 1 thru 7:

1 = Solid (round ends)

2 = Long Dash

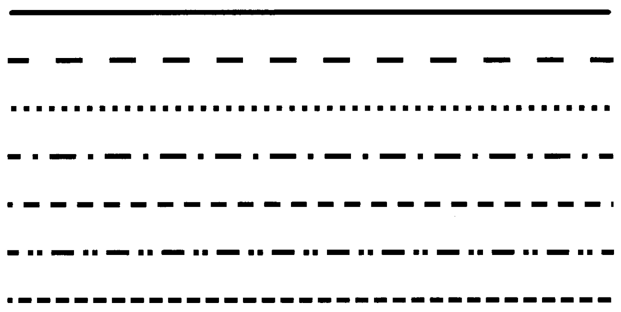
3 = Dotted

4 = Dash Dotted

5 = Medium Dash

6 = Dash With Two Dots

7 = Short Dash



Graphics Markers

The IBM Extended Monochrome Graphics Display device driver supports six Graphics Marker types, selected with type indexes 1 thru 6:

1 = Dot	■
2 = Cross	+
3 = Star	✱
4 = Square	□
5 = X	✕
6 = Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following table shows the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios.

Graphics Marker Sizes		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	224	299
2	416	555
3	608	811
4	800	1067
5	992	1323

Graphics Text

Graphics Text fonts are selected by index number. Index number 1 is reserved for the default font. All other index numbers are derived from an alphabetical list of font files that end with the .9X20 extension and reside in **/etc/vtm.** For example, the first file in the alphabetical list would be index number 2, the second would be index number 3 and so on.

Graphics Text can be rotated on 0, 90, 180, and 270 degree baselines. The IBM Extended Monochrome Graphics Display device driver supports five Graphics Text sizes. These sizes are listed below in NDC units:

Graphics Text Sizes Preserve Aspect Ratio				
Index	Character Height	Character Width	Cell Height	Cell Width
1	512	256	640	288
2	1024	512	1280	576
3	1536	768	1920	864
4	2048	1024	2560	1152
5	2560	1280	3200	1440

Graphics Text Sizes Non-Preserve Aspect Ratio				
Index	Character Height	Character Width	Cell Height	Cell Width
1	683	256	854	288
2	1366	512	1707	576
3	2048	768	2560	864
4	2731	1024	3414	1152
5	3414	1280	4267	1440

Filled Areas

Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

1=Right narrow diagonal

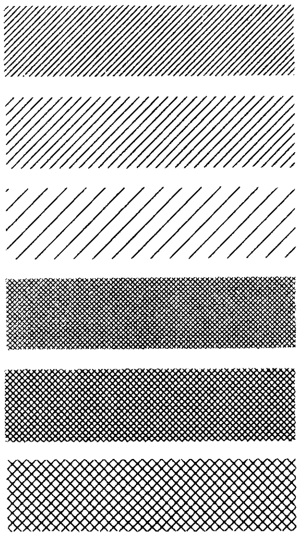
2=Right medium diagonal

3=Right wide diagonal

4=Narrow diagonal crosshatch

5=Medium diagonal crosshatch

6=Wide diagonal crosshatch



Colors

In graphics mode, there are two available colors that can be used to display graphics primitives. Color index 0 is displayed as black and color index 1 is displayed as white. These colors cannot be redefined.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available on the IBM Extended Monochrome Graphics Display device driver:

Fonts:	Font index 1 is the standard device font, fonts 2 through 6 are reserved for VDI, and font indexes 7 and upwards are taken from an alphabetical list of files that end with the .9X20 extension and reside in **/etc/vtm.**
Sizes:	One Size
Features:	Underlining Overstrike Mode Superscript and Subscript Line Spacing

Request Locator

When locator is invoked, a tracking cross appears on the display at the initial locator position. The cross can be moved with the IBM RT PC Mouse or by pressing one of the four arrow keys on the keyboard.

Initially, the cross moves in large increments. Pressing the **Insert** key toggles the distance between large movements and small movements. When the cross is at the desired location, the point can be selected by pressing any alpha key on the keyboard. This causes the coordinates of the point to be transmitted back to the user program.

If desired, the device will perform an inking function. When the locator is terminated, a line from the initial position to the desired position is drawn honoring the current line attributes of the output echo device, such as color and line style. Also, the output echo device performs rubberbanding if desired.

Two types of rubberbanding are supported, lines and boxes. If rubberbanding lines are desired, then a line will be drawn from the initial locator position to the current position of the graphics cursor. The line changes dynamically as the cursor is moved. When the locator is terminated, the line is removed.

If rubberband rectangle is specified, a rectangle is displayed with one corner at the initial locator position and the opposite at the current position of the graphics cursor. The rectangle changes dynamically as the cursor is moved. When the locator is terminated, the rectangle is removed from the display.

Request Choice

The function keys **F1** thru **F12** on the keyboard, in conjunction with the **SHIFT, CTRL,** and **ALT** keys, are used to enter choice input. Thus, a total of 48 different inputs can be generated. Pressing keys **F1** thru **F12** alone will generate inputs 1 thru 12, respectively. If the **SHIFT** key is held down, while pressing keys **F1** thru **F12,** inputs 13 thru 24 will be generated. If the **CTRL** key is held down, and the twelve keys are pressed, inputs 25 thru 36 will be generated. If the **ALT** key is held down, and the twelve keys are pressed, inputs 37 thru 48 will be generated.

Request String

The keyboard is used to enter strings. The string is terminated by the **Enter** key.

Cursor Addressable Text

Cursor addressable text is supported. The device must be in Cursor Addressing Mode before it can perform any cursor control functions. To display graphics primitives, the device must be removed from Cursor Addressing Mode. The Reverse Video attribute is supported.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM Extended Monochrome Graphics Display. If your application calls a routine that is not supported, an error will occur (error code –5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Update Workstation.

Paging Routines

- Copy Page
- Inquire Page
- Set Page.

Pel Routines

- Copy Pels
- Get Pels
- Put Pels.

Cursor Control Routines

- Cursor Down
- Cursor Home
- Cursor Left
- Cursor Right
- Cursor Up
- Direct Cursor Address
- Enter Cursor Addressing Mode
- Erase to End of Line
- Erase to End of Screen
- Exit Cursor Addressing Mode
- Inquire Addressable Character Cells
- Inquire Current Cursor Text Address
- Output Cursor Addressable Text
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index.

General Graphics Routines

- Display Graphic Input Cursor
- Inquire Color Representation
- Remove Graphic Input Cursor

- Set Background Color Index
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability

- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Choice (request mode)
- Input Locator (request mode)
- Input String (request mode)
- Input String (sample mode)
- Read Cursor Movement Keys
- Set Line Edit Characters.

Error Handling

- Inquire Error.

IBM Enhanced Graphics Adapter

Filename: vdiega

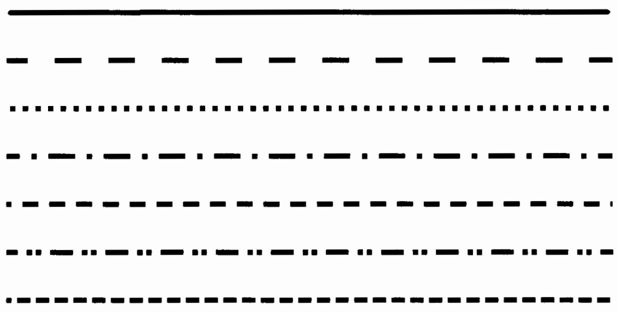
Features Supported

The following text describes each of the features of the IBM Enhanced Graphics Adapter with an Enhanced Color Display or a Monochrome Monitor. These features apply equally to both the display or monitor, except where noted in the individual feature descriptions.

Polylines

Lines and arcs can be drawn on the IBM Enhanced Graphics Adapter, with an Enhanced Color Display or Monochrome Monitor, with one of seven line styles, selected with style indexes 1 thru 7:

1=Solid (round ends)
2=Long Dash
3=Dotted
4=Dash Dotted
5=Medium Dash
6=Dash With Two Dots
7=Short Dash



Graphics Markers

The IBM Enhanced Graphics Adapter, with either the Enhanced Color Display or the Monochrome Monitor, supports six Graphics Marker types, selected with type indexes 1 thru 6:

1=Dot	■
2=Cross	+
3=Star	✱
4=Square	□
5=X	✕
6=Diamond	◇

Each type of marker can be drawn in one of five sizes, selected with size indexes 1 thru 5. The following table shows the NDC units for the five Graphics Marker sizes, in both Preserve and Non-Preserve Aspect Ratios.

Graphics Marker Sizes		
Index	Preserve Aspect Ratio	Non-Preserve Aspect Ratio
1	456	656
2	846	1218
3	1236	1779
4	1626	2341
5	2016	2903

Graphics Text

Graphics Text fonts are selected by index number. Index number 1 is reserved for the default font. All other index numbers are derived from an alphabetical list of font files that end with the .9X20 extension and reside in **/etc/vtm.** For example, the first file in the alphabetical list would be index number 2, the second would be index number 3 and so on.

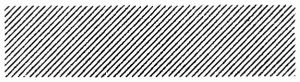
Graphics Text can be rotated on 0, 90, 180, and 270 degree baselines. The IBM Enhanced Graphics Adapter device driver supports five Graphics Text sizes. These sizes are listed below in NDC units:

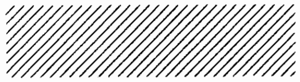
Graphics Text Sizes Preserve Aspect Ratio Mode				
Index	Character Height	Character Width	Cell Height	Cell Width
1	716	359	911	410
2	1431	717	1821	820
3	2146	1076	2731	1229
4	2861	1434	3641	1639
5	3576	1792	4552	2048

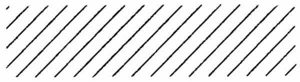
Graphics Text Sizes Non-Preserve Aspect Ratio Mode				
Index	Character Height	Character Width	Cell Height	Cell Width
1	1030	359	1311	410
2	2060	717	2622	820
3	3090	1076	3933	1229
4	4120	1434	5243	1639
5	5150	1792	6554	2048

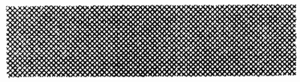
Filled Areas

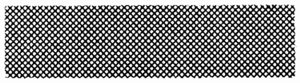
Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. Hollow, Solid, and Hatch interior styles are supported. The Pattern interior style is mapped to the Hatch interior style. There are six Hatch interior styles, as follows:

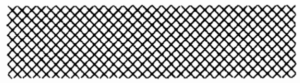
1 = Right narrow diagonal 

2 = Right medium diagonal 

3 = Right wide diagonal 

4 = Narrow diagonal crosshatch 

5 = Medium diagonal crosshatch 

6 = Wide diagonal crosshatch 

Colors

The device driver for the IBM Enhanced Graphics Adapter with a Monochrome Monitor supports four color attributes. They are:

0 = Background
1 = Video
2 = Video Blink
3 = Bold Video

The four color attributes for the Monochrome Monitor cannot be redefined.

The device driver for the IBM Enhanced Graphics Adapter with Enhanced Color Display supports either four or sixteen colors, depending on the amount of memory available on the Enhanced Graphics Adapter card.

In Four Color Mode (with 64K of memory on the card) the Enhanced Color Display will support the following color attributes:

0=Black
1=White
2=Red
3=Green

In Sixteen Color Mode (with either 128K or 256K of memory on the card) the Enhanced Color Display will support the following color attributes:

0=Black
1=White
2=Red
3=Green
4=Blue
5=Yellow
6=Cyan
7=Magenta
8-15=White

The color indexes for the Enhanced Color Display can be redefined. Each of the Red, Green, and Blue components can have 0, 333, 666, or 1000 values. This enables you to have 64 colors. The Set Color Representation function will set only an individual color index and not establish a palette.

Alpha Text

Alpha Text can be positioned anywhere on the output page. The following text capabilities are available on the IBM Enhanced Graphics Adapter with the Monochrome Monitor or Enhanced Color Display:

Fonts:	Only one standard font called by index 1.
Sizes:	One Size

Features:	Underlining
	Overstrike Mode
	Superscript and Subscript
	Line Spacing
	Color (Enhanced Color Display only)

Request Locator

When locator is invoked, a tracking cross appears on the display at the initial locator position. The cross can be moved with the IBM RT PC Mouse or by pressing one of the four arrow keys on the keyboard.

Initially, the cross moves in large increments. Pressing the **Insert** key toggles the distance between large movements and small movements. When the cross is at the desired location, the point can be selected by pressing any alpha key on the keyboard. This causes the coordinates of the point to be transmitted back to the user program.

If desired, the device will perform an inking function. When the locator is terminated, a line from the initial position to the desired position is drawn honoring the current line attributes of the output echo device, such as color and line style. Also, the output echo device performs rubberbanding if desired.

Two types of rubberbanding are supported, lines and boxes. If rubberbanding lines are desired, then a line will be drawn from the initial locator position to the current position of the graphics cursor. The line changes dynamically as the cursor is moved. When the locator is terminated, the line is removed.

If rubberband rectangle is specified, a rectangle is displayed with one corner at the initial locator position and the opposite at the current position of the graphics cursor. The rectangle changes dynamically as the cursor is moved. When the locator is terminated, the rectangle is removed from the display.

Request Choice

The function keys **F1** thru **F12** on the keyboard, in conjunction with the **SHIFT, CTRL,** and **ALT** keys, are used to enter choice input. Thus, a total of 48 different inputs can be generated. Pressing keys **F1** thru **F12** alone will generate inputs 1 thru 12, respectively. If the **SHIFT** key is held down, while pressing keys **F1** thru **F12,** inputs 13 thru 24 will be generated. If the **CTRL** key is held down, and the twelve keys are pressed, inputs 25 thru 36 will be generated. If the **ALT** key is held down, and the twelve keys are pressed, inputs 37 thru 48 will be generated.

Request String

The keyboard is used to enter strings. The string is terminated by the **Enter** key.

Cursor Addressable Text

Cursor addressable text is supported. The device must be in Cursor Addressing Mode before it can perform any cursor control functions. To display graphics primitives, the device must be removed from Cursor Addressing Mode. The Reverse Video, Blink, and Bold intensity attributes are supported.

Device Specific Information

The following tables show the screen modes that are available for the Enhanced Color Display and for the Monochrome Monitor.

Enhanced Color Display Screen Modes		
Screen Mode	Screen Size	Number of Pages
Cursor Text	80X25	4 - 64K 8 - otherwise
Graphics Text	640X350	1 - 64K (4 color) 1 - 128K (16 color) 2 - 256K (16 color)

Monochrome Monitor Screen Modes		
Screen Mode	Screen Size	Number of Pages
Cursor Text	80X25	4 - 64K 8 - otherwise
Graphics Text	640X350	1 - 64K (4 color) 1 - 128K (4 color) 2 - 256K (4 color)

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM Enhanced Graphics Adapter, with Enhanced Color Display or Monochrome Monitor. If your application calls a routine that is not supported, an error will occur (error code –5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Open Workstation
- Update Workstation.

Paging Routines

- Copy Page
- Inquire Page
- Set Page.

Pel Routines

- Copy Pels
- Get Pels
- Put Pels.

Cursor Control Routines

- Cursor Down
- Cursor Home
- Cursor Left
- Cursor Right
- Cursor Up
- Direct Cursor Address
- Enter Cursor Addressing Mode
- Erase to End of Line
- Erase to End of Screen

- Exit Cursor Addressing Mode
- Inquire Addressable Character Cells
- Inquire Current Cursor Text Address
- Output Cursor Addressable Text
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index.

General Graphics Routines

- Display Graphic Input Cursor
- Inquire Color Representation
- Remove Graphic Input Cursor
- Set Background Color Index
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width
- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Choice (request mode)
- Input Locator (request mode)
- Input String (request mode)
- Input String (sample mode)
- Read Cursor Movement Keys
- Set Line Edit Characters.

Error Handling

- Inquire Error.

IBM Virtual Device Metafile (VDM) Driver

Filename: vdimeta

Features Supported

The following text describes each of the IBM Virtual Device Metafile features that are supported by this device driver.

Polylines

The IBM Virtual Device Metafile Driver supports 32,767 line styles.

Graphics Markers

The IBM Virtual Device Metafile Driver supports 32,767 types of Graphics Markers.

Graphics Text

The IBM Virtual Device Metafile Driver supports 32,767 character sizes and 32,767 character rotations.

Filled Areas

Filled areas, bars, pie slices and circles are displayed using the current fill area attributes of color, interior style, and style index. The interior style of Hollow, Solid, and Hatch are all supported by this device driver.

Colors

The IBM Virtual Device Metafile Driver supports 256 color indexes. Each color index represents some combination of the three primary colors. That is, a color is defined by setting the intensity levels for the R (red), G (green), and B (blue) color components. Each intensity level must be set to some number from zero to 1000, inclusive.

To assign another color to an index, re-define the R, G, and B color intensity levels. You can then create both the more common or default colors and those not usually found on graphics devices, such as brown or orange. All colors will, however, be visible only on devices that allow color definition.

Alpha Text

The Metafile Driver supports Alpha Text. The following text capabilities are available.

Font:	Supports up to 32,767 font indexes.
Features:	Underlining Overstrike Mode Superscript and Subscript Line Spacing

Cursor Addressable Text

Cursor addressable text is supported. The device must be in Cursor Addressing Mode before it can perform any cursor control functions. To display graphics primitives, the device must be removed from Cursor Addressing Mode. The Reverse Video, Blink, Color (expanded palette), and Bold intensity attributes are supported.

Raster Writing Modes

Raster writing modes define how pixels are set in the bit map. The Metafile Driver supports all 16 raster writing modes (Boolean operations between source and destination).

Device Specific Information

If a filename is not specified, the output of an application program that uses this device driver to generate a metafile will be directed to a default file called **METAFILE.DAT.** To select an alternate filename, you must export the name of the target file to the system environment, as follows:

```
METAOUTPUT=filename                                  (sh shell)
export METAOUTPUT
```

(or)

```
setenv METAOUTPUT filename                           (csh shell)
```

Refer to the *IBM RT PC Graphical File System Programmer's/User's Guide* for more information about setting environmental parameters associated with a metafile.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM Virtual Device Metafile. If your application calls a routine that is not supported, an error will occur (error code –5000).

Workstation Control Routines

- Application Data
- Clear Workstation
- Close Workstation
- Hardcopy
- Message
- Open Workstation
- Update Workstation.

Cursor Control Routines

- Cursor Down
- Cursor Home
- Cursor Left
- Cursor Right
- Cursor Up
- Direct Cursor Address

- Enter Cursor Addressing Mode
- Erase to End of Line
- Erase to End of Screen
- Exit Cursor Addressing Mode
- Inquire Addressable Character Cells
- Inquire Current Cursor Text Address
- Output Cursor Addressable Text
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index.

General Graphics Routines

- Display Graphic Input Cursor
- Inquire Color Representation
- Remove Graphic Input Cursor
- Set Background Color Index
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width

- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Choice (request mode)
- Input Choice (sample mode)
- Input Locator (request mode)
- Input Locator (sample mode)
- Input String (request mode)
- Input String (sample mode)
- Input Valuator (request mode)

- Input Valuator (sample mode)
- Read Cursor Movement Keys
- Set Line Edit Characters.

Error Handling

- Inquire Error.

IBM RT PC Grafstation Driver

Filename: vdigst

Features Supported

The IBM RT PC Grafstation Driver forms an interface between the IBM RT PC and the IBM PC, PC XT, and PC AT. This interface is capable of passing information about any of the features that are supported by the IBM RT PC Graphics Development Toolkit. Thus, the features that are supported by this device driver are limited only to the set of features supported by the IBM PC, PC XT, and PC AT and their graphics peripherals.

Device Specific Information

The IBM RT PC Grafstation Driver is a part of the IBM RT PC Graphics Development Toolkit. However, in order for this device driver to operate properly, additional software must be installed on your IBM PC, PC XT, or PC AT.

Routine Summary

The following list shows the Toolkit routines that are supported by the device driver for the IBM RT PC Grafstation Driver. If your application calls a routine that is not supported, an error will occur (error code -5000).

Workstation Control Routines

- Clear Workstation
- Close Workstation
- Hardcopy
- Open Workstation
- Update Workstation.

Cursor Control Routines

- Cursor Down
- Cursor Home
- Cursor Left
- Cursor Right

- Cursor Up
- Direct Cursor Address
- Enter Cursor Addressing Mode
- Erase to End of Line
- Erase to End of Screen
- Exit Cursor Addressing Mode
- Inquire Addressable Character Cells
- Inquire Current Cursor Text Address
- Output Cursor Addressable Text
- Reverse Video Off
- Reverse Video On
- Set Cursor Text Attributes
- Set Cursor Text Color Index.

General Graphics Routines

- Display Graphic Input Cursor
- Inquire Color Representation
- Remove Graphic Input Cursor
- Set Background Color Index
- Set Color Representation
- Set Writing Mode.

Graphics Primitives

- Inquire Cell Array
- Inquire Current Fill Area Attributes
- Inquire Current Polyline Attributes
- Inquire Current Polymarker Attributes
- Output Arc
- Output Bar
- Output Cell Array
- Output Circle
- Output Filled Area
- Output Pie Slice
- Output Polyline
- Output Polymarker
- Set Fill Color Index
- Set Fill Interior Style
- Set Fill Style Index
- Set Polyline Color Index
- Set Polyline Line Type
- Set Polyline Line Width

- Set Polymarker Color Index
- Set Polymarker Height
- Set Polymarker Type.

Graphics Text Routines

- Inquire Current Graphic Text Attributes
- Output Graphic Text
- Set Character Height
- Set Graphic Text Alignment
- Set Graphic Text Color Index
- Set Graphic Text Font
- Set Graphic Text String Baseline Rotation.

Alpha Text Routines

- Inquire Alpha Text Capabilities
- Inquire Alpha Text Cell Location
- Inquire Alpha Text Font Capability
- Inquire Alpha Text Position
- Inquire Alpha Text String Length
- Output Alpha Text
- Set Alpha Text Color Index
- Set Alpha Text Font and Size
- Set Alpha Text Line Spacing
- Set Alpha Text Overstrike Mode
- Set Alpha Text Pass Through Mode
- Set Alpha Text Position
- Set Alpha Text Quality
- Set Alpha Text Subscript Superscript Mode
- Set Alpha Text Underline Mode.

Input Routines

- Input Choice (request mode)
- Input Choice (sample mode)
- Input Locator (request mode)
- Input Locator (sample mode)
- Input String (request mode)
- Input String (sample mode)
- Input Valuator (request mode)

- Input Valuator (sample mode)
- Read Cursor Movement Keys
- Set Line Edit Characters.

Error Handling

- Inquire Error.

Appendix D. Error Codes

This appendix lists the error codes that are generated by the Graphics Development Toolkit.

The Graphics Development Toolkit routines always return a *status code,* whether or not the requested operation was successful. If this status code is –1, indicating that an error has occurred, the Inquire Error routine must be used to obtain the actual error code.

Determining Toolkit Errors

The Graphics Development Toolkit error codes are separated into three types of codes:

- General Toolkit error codes
- Specific Toolkit error codes
- Special Toolkit error codes.

To determine an error code, first locate the general error code. Then, add the specific error code to it. For example, an error code of –508 means:

	–500	Error while opening a file.
+		
	–8	Insufficient memory for requested operation.
=	–508	The actual error code.

General Toolkit Error Codes

The following list contains a brief explanation of general Toolkit error codes that may be returned by the Inquire Error routine.

Code	Meaning
-200	Error during the file connect operation.
-300	Error during the file disconnect operation.
-400	Error during the copy file descriptor operation.
-500	Error while opening a file.
-600	Error while closing a file.
-700	Error while reading a file in wait mode.
-800	Error while writing to a file wait mode.
-1200	Error during seek operation.
-1400	Error during file delete operation.
-1800	Error during dashed line output.
-1900	Error during marker output.
-2000	Error during text alignment output.
-2100	Error during polygon output.
-2200	Error during bar output.
-2300	Error during arc output.
-2400	Error during pie output.
-2500	Error during circle output.
-2600	Error initializing input device.
-2700	Error moving cursor during GIN.
-2800	Error terminating GIN function.

Specific Toolkit Error Codes

The following list contains a brief explanation of specific Toolkit error codes that may be returned by the Inquire Error routine. This list is followed by additional explanation and suggestions about how to respond to the error code.

Code	**Meaning**
-5	Access denied.
-8	Insufficient memory for requested operation.
-79	Device is busy.
-80	Device or hardware not present.
-81	All (driver) slots used.
-82	Toolkit can't start.
-84	Driver can't start.
-86	Slot (driver) already open.
-89	No driver file.
-90	Unknown driver, (driver file does not exist).
-92	Illegal device handle.

–5 **Access denied:**

Cause: A request for Monitor Mode was denied. The requested driver is only accessible from the system console.

Action: Check that you are requesting the correct device. Use the system console if necessary.

Cause: Unable to create lock file for device driver. When an application selects a device for output, a lock file is created to prevent multiple applications from mixing their output on the same device. The device driver was unable to create this lock file.

Action: Check the directory for permission to create a file for that port.

If the AIX Extended Services programs are installed on the system, lock files are located in directory **/usr/spool/uucp**. If the Extended Services programs are not installed, lock files are located in **/usr/lpp/vdi/locks**.

–8 **Insufficient memory for requested operation:**

Cause: This error means that you do not have enough system memory available to perform the requested operation.

Action: Add more memory to the system.

–79 **Device is busy:**

Cause: This error is generated when a lock file has been created for the device you are attempting to use. A lock file is created each time you perform graphics to a device that is not assigned to the logical device /dev/tty. The naming of the lock file has the form:

```
LCK..ttyxx
```

Where xx is the identification of the logical device being requested.

If the AIX Extended Services programs have been installed on the system, the lock file is located in directory **/usr/spool/uucp**. If the Extended Services programs have not been installed on the system, the lock file will be located in directory **/usr/lpp/vdi/locks**.

Action:
a) If the lock file was created by another graphics application, then you must wait for that other graphics application to release control of the device.

b) If the file was a remnant of a previous program error, delete the lock file and run your program again.

-80 **Device or hardware not present:**

Cause: The physical device is not attached to the IBM RT PC.

Action: Attach the physical device and restart your application.

-81 **All (driver) slots used:**

Cause: Each application is limited to having no more than eight graphics devices open simultaneously.

Action: Close one of the open graphics devices.

-82 **Toolkit can't start:**

Cause: This error message is generated when the Inter Process Communication (IPC) of the system is full. This occurs when you have exceeded the number of system shared memory areas or semaphores configured for your system. Two shared memory areas and one semaphore group are used for each graphics application. The shared memory areas are 512K and 362 bytes. The semaphore group contains eight semaphores.

Action:
a) Make sure that shared memory areas are at least 512K in size.
b) Rebuild the AIX Operating System with more semaphores and/or shared memory areas.
c) Remove any unused shared memory areas with the **ipcrm** command.
d) Kill any unused running applications.

-84 **Driver can't start:**

Cause: The **fork** command, used to start the physical graphics device driver as a new process, failed.

Action:
a) Wait until the total number of executing processes is less than the system-imposed process limit.
b) Wait until the total number of executing processes for a single user is less than the system-imposed process limit.
c) Ensure that there is enough paging space or physical memory for the process. If necessary, increase the paging area or add more physical memory.

-86 **Slot (driver) already open:**

Cause: Trying to open a device that is already open–an application can only open a driver once.

Action: a) Ensure that you are requesting the correct device. Refer to "Setting Environmental Parameters" in Appendix A for information about assigning logical device names to graphics devices.

b) Ensure that you have spelled the device name correctly.

c) Determine which is the existing device **handle** and either use this identifier or close and re-open the device.

-89 **No driver file:**

Cause: The name of the logical device cannot be found.

Action: Ensure that the logical device name is defined in the current shell environment.

-90 **Unknown driver, (driver file does not exist):**

Cause: Within the Toolkit, graphics devices are referred to by logical device names, (e.g. DISPLAY, PRINTER, PLOTTER, etc.). These names must be associated with physical graphics device drivers.

Action: a) Check that you are requesting the correct device. Refer to "Setting Environmental Parameters" in Appendix A for information about assigning logical device names to graphics devices.

b) Check that you have spelled the device name correctly.

c) Check that the driver you are requesting is located in the VDIPATH specified.

-92 **Illegal device handle:**

Cause: An incorrect device handle has been used.

Action: Check the device handle value returned by the Open Workstation routine.

Special Toolkit Error Codes

The following list contains a brief explanation of special Toolkit error codes that may be returned by the Inquire Error routine. This list is followed by additional explanation and suggestions about how to respond to the error code.

Code	**Meaning**
–3095	Invalid page.
–5000	Function not supported by current device driver.
〈–24000	Communication port errors.

–3095 Invalid page.

Cause: When calling the Copy Page routine, an invalid page number was returned.

Action: Compare both input and output parameters for the Inquire Page routine.

Cause: When calling the Inquire Current Cursor Text Address, current page is in graphics mode.

Action: No action required. This can be used to determine whether the current page is in graphics or cursor text mode.

–5000 Function not supported by current device driver.

Cause: The function requested is not available on the specified device driver.

Action:

a) Check that you have requested the correct function.

b) Check the device-dependent information pages to confirm that the requested feature is supported.

c) Review the available functions list to determine if the desired information can be obtained through the use of another function.

<-24000 Communication port errors:

Cause: Device drivers require the use of communication ports. In attempting to open a communication port for graphics output, an error may be generated if the device configuration does not match the port configuration.

Action: a) Check the port configuration.

b) Confirm that the device matches the port configuration.

c) Ensure that the port is configured for read and write permission.

Advanced Programming Notes

The following text contains a few brief notes about possible error conditions that can occur during the interaction between IBM RT PC Graphics Development Toolkit and device drivers. This information is intended for the advanced programmer and assumes a high degree of expertise with the IBM RT PC AIX Operating System.

The Graphics Development Toolkit drivers support **signal**, the IBM RT PC AIX Operating System error handling mechanism.

In certain software/hardware error conditions, such as an illegal memory reference or an illegal CPU instruction, the AIX Operating System sends a signal to a process to tell it that an error has occurred. By default, the operating system terminates the process as a result of the error.

When an application that uses the Toolkit drivers receives the signal, the driver system is shut down. Before exiting, the I/O channels that were being used are reset.

If the application writer intercepts signals before the first Open Workstation call to the Toolkit drivers, the application's signal handler will be called after the Toolkit drivers signal handler has shut down the driver system. If the application intercepts signals after an Open Workstation call to the Toolkit drivers, the application's signal handler is called first. The application must call the Toolkit drivers signal handler after it is through processing the signal. See the description of the **signal** system call in the *IBM RT PC AIX Operating System Technical Reference.*

Special Error Handling Messages

In the event of an abnormal termination of the Toolkit drivers, one of the following error messages may be displayed.

SIGQUIT	–Received quit signal: Application Terminated
SIGILL	–Illegal Instruction: Application Terminated
SIGTRAP	–Trace Trap: Application Terminated
SIGIOT	–IOT Instruction: Application Terminated
SIGEMT	–EMT Instruction: Application Terminated
SIGBUS	–Bus Error: Application Terminated
SIGSEGV	–Segmentation Violation: Application Terminated
SIGSYS	–Bad Argument to System Call: Application Terminated

If abnormal termination is caused by any other signal, only the words "Application Terminated" will appear on the screen.

Note: Floating Point Exceptions (SIGFPE) are caught by Toolkit and ignored. The program will not terminate abnormally as a result of floating point exceptions. The programmer may choose to set up a signal handler to handle these exceptions if they are important to the application.

Glossary

ADE. ASCII decimal equivalents are decimal numbers used in code to represent ASCII characters. For example, the integer 65 represent the letter "A" and the integer 66 represents the letter "B".

argument. One of the independent variables that the action or output of a routine depends on. Arguments are enclosed in parentheses in the routine call.

array. A set of related elements (data) arranged in a specific pattern.

ASCII. ASCII stands for American Standard Code for Information Interchange. This standard for data transmission assigns individual 7-bit codes to represent each of a specific set of 128 numerals, letters, and special controls.

aspects of primitives. Ways in which the appearance of a primitive can vary. Aspects are controlled directly by primitive attributes.

attribute functions. Primitive attributes affect the appearance of objects created with primitive routines. (Examples: character height, line style)

binding. Language binding refers to the exact calling syntax and data type specification for arguments to be used when calling Toolkit routines from a specific programming language.

Cartesian coordinate system. Coordinate system composed of an X-axis (horizontal) increasing positively towards the right, and a Y-axis (vertical) increasing positively upwards. The axes are positioned at right angles, and the point of intersection is the origin (0.,0.). The position of any point is defined by displacement from the origin along first the X-axis and then the Y-axis.

cell array. Toolkit output primitive consisting of a rectangular grid of equal size rectangular cells, each having a single color. These cells may not change one-to-one with frame buffer pels.

clipping. When you set a window in the Normalized Device Coordinate, part of an object may lie outside the window. In this case, the part lying outside the window will be clipped; that is, it will not be displayed on the viewport.

color map. Table designed to provide a range of colors by defining different mixtures of the color components. A desired color is referenced by its assigned number. The identifying numbers with their assigned colors are called the color map. Changing colors assigned to the identifying number changes the map.

color table. Workstation-dependent table in which the entries specify the values of the red, green, and blue intensities defining a particular color.

control functions. These facilities allow you to exercise control over certain aspects of the system and the display device. The Toolkit provides a means to access the nonstandard capabilities of your display device through an escape mechanism invoked with the escape routines.

coordinate graphics. Computer graphic in which display images are generated from display commands and coordinate data.

coordinate scaling. Coordinate scaling transforms points from one space to another. In the Toolkit, all point coordinates must be specified in Normalized Device Coordinates with values between 0 and 32,767. These coordinates are then scaled into values which are appropriate for your graphics device.

default. A value assigned to a parameter by the Toolkit, and used when you do not specify a value.

DC. See **Device Coordinate.**

DC unit. The Device Coordinate (DC) unit is a unit of measure for the physical space represented by the display surface. The Toolkit translates NDC units into DC units and vice versa.

Device Coordinate. A coordinate expressed in a coordinate system that is device-dependent.

device driver. Device-dependent software that generates instructions specifying items to be drawn on the display surface from the invocations of the Toolkit.

device-independent. The ability to be used on more than one type of graphics display device.

device space. The space defined by the addressable points of a display device.

display device. A device (for example, refresh display, storage tube display, plotter) on which display images can be represented.

display surface. In a display device, that medium on which display images may appear.

echo. The immediate notification of the current value provided by an input device to the operator at the display console.

echo handle. The device handle used to display the graphic cursor on an echo device.

escape. Routines within the Toolkit that are the only access to implementation-dependent or device-dependent support for nonstandard Routines other than graphic output.

fill area. A Toolkit output primitive consisting of a polygon (closed boundary) which can be hollow or can be filled with a uniform color, a pattern, or a hatch style.

GDP. See **Generalized Drawing Primitive.**

Generalized Drawing Primitive. The Generalized Drawing Primitive (GDP) is a display element (output primitive) used to address special geometrical workstation capabilities such as curve drawing.

graphics primitives. Graphics primitives are the basic graphics operations performed by the Toolkit; for example, drawing lines, markers, and text strings.

handle. A number returned when the workstation is opened. This number is unique for each device that is open. Any Toolkit routine directed at a workstation must use a unique handle.

host-independent. Capable of running on a number of operating systems.

input functions. The Toolkit allows you to obtain the value of an NDC coordinate point from an interactive graphics device. The method for specifying the point is device-dependent.

inquiry functions. The Toolkit provides inquiry facilities that allow your program to determine the present state of the system. You may determine the current value of the following:

- Primitive attributes
- Device capabilities
- Device state.

locator device. A Toolkit logical input device providing a position in Normalized Device Coordinates.

NDC. See **Normalized Device Coordinates.**

NDC unit. The Normalized Device Coordinate (NDC) unit is a unit of measure for the virtual space through which a graphics application program passes graphics information to a device. The Toolkit translates NDC units into DC units and vice versa.

Normalized Device Coordinates. The Toolkit introduces the concept of a Normalized Device Coordinate (NDC) unit in which the full extent of the device axes are assigned values between 0 and 32,767. This convention provides improved device independence for a graphics system by allowing the viewing operations to be carried out without regard for device specifics. The NDC coordinates are then converted to specific Device Coordinates.

null-terminated string. A string is a one-dimensional array or list of characters. The end of a string is indicated by the ASCII NULL character (ADE 0).

output primitives. The graphical world which the programmer describes consists of one or more objects. These are created and modified by invocations of graphic primitive routines provided by the Toolkit. These routines describe polylines, polymarkers, text strings, pel arrays, fill areas, and Generalized Drawing Primitives. The appearance of output primitives is affected by the values of primitive attributes.

pel. The term pel, also known as a pixel, refers to a "picture element"; the smallest element of a display surface that can be independently assigned a color or intensity.

pixel. See **pel.**

polyline. A Toolkit output primitive consisting of a set of connected lines.

polymarker. A Toolkit output primitive consisting of a series of marker symbols drawn at specified points.

raster. A field of closely spaced lines on the face of a video terminal that defines an image. The spacing between raster lines defines the resolution of a display.

RGB. A method for defining one color in terms of the intensity of each of the three primary colors. With this method, percentages of red, green, and blue are "added" to produce colors. For example, a color intensity of 100% red plus 100% green plus 100% blue equals white.

Toolkit. The Toolkit is a host-independent and device-independent graphics subsystem that serves as an environment for graphics applications as well as application development.

transformation. The changing of objects from one coordinate space to another.

Virtual Device Interface. The Virtual Device Interface (VDI) is a standard interface between device-dependent and device-independent code in a graphics environment. VDI makes all device drivers appear identical to the calling program.

workstation. The Toolkit is based on the concept of abstract graphical workstations which provide the logical interface through which the applications program controls physical devices.

Index

IBM

The IBM RT PC Family

Reader's Comment Form

IBM RT PC Graphics Development Toolkit SV21-8058

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact the authorized IBM RT PC dealer in your area.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

Tape

Please Do Not Staple

Tape

IBM

The IBM RT PC Family

Reader's Comment Form

IBM RT PC Graphics Development Toolkit

SV21-8058

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact the authorized IBM RT PC dealer in your area.

Comments:

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

Tape

Please Do Not Staple

Tape

IBM RT PC GRAPHICS DEVELOPMENT TOOLKIT

Book Title

SV21-8058

Order No.

Book Evaluation Form

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y N Is the purpose of this book clear?

Y N Is the table of contents helpful?

Y N Is the index complete?

Y N Are the chapter titles and other headings meaningful?

Y N Is the information organized appropriately?

Y N Is the information accurate?

Y N Is the information complete?

Y N Is only necessary information included?

Y N Does the book refer you to the appropriate places for more information?

Y N Are terms defined clearly?

Y N Are terms used consistently?

Y N Are the abbreviations and acronyms understandable?

Y N Are the examples clear?

Y N Are examples provided where they are needed?

Y N Are the illustrations clear?

Y N Is the format of the book (shape, size, color) effective?

Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

Optional Information

Your name

Company name

Street address

City, State, ZIP

No postage necessary if mailed in the U.S.A.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Fold and tape

Fold and tape

Cut or Fold Along Line

Tape

Please Do Not Staple

Tape

International Business
Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758

Printed in the
United States of America

59X8554