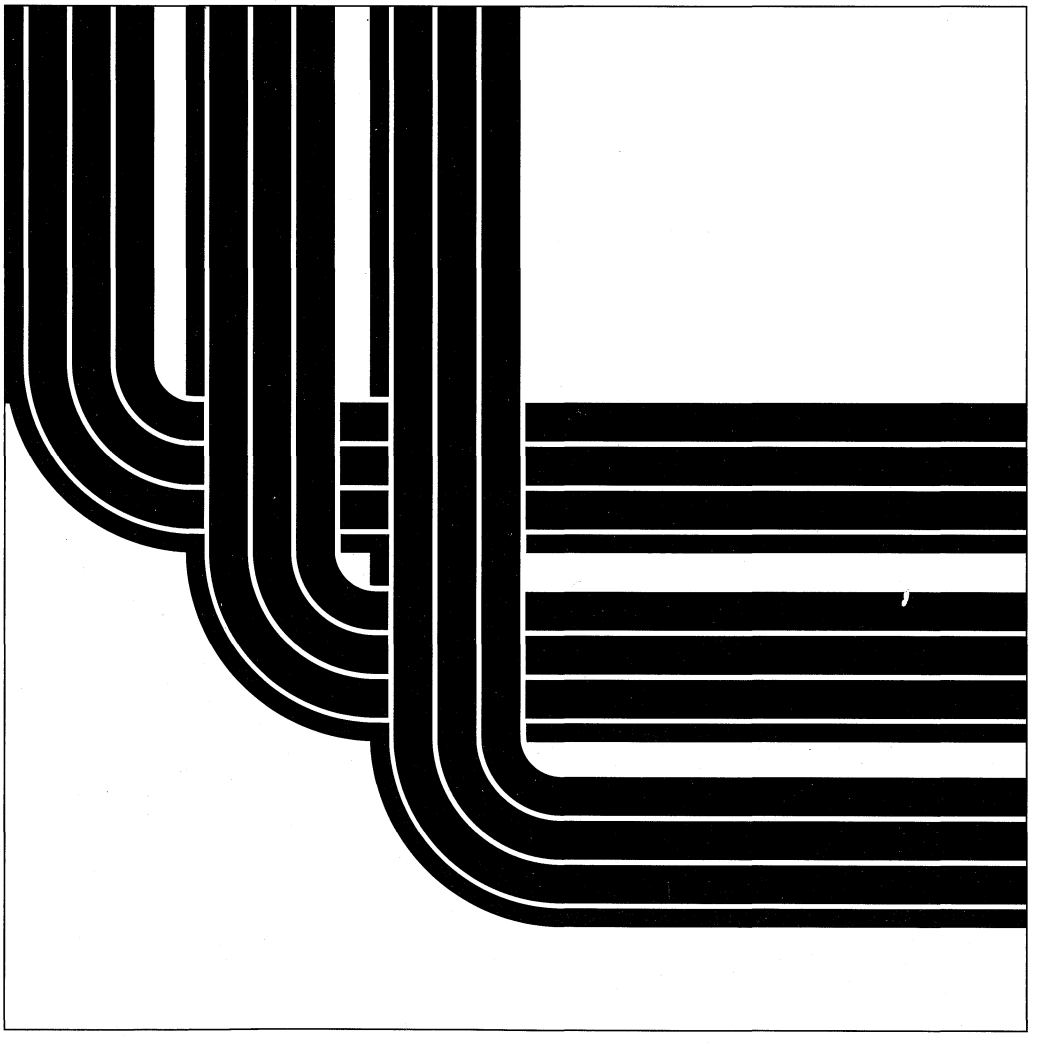
Application System/400

SC41-0536-00

Programming: GDDM Programming Guide

Version 2



Application Development

Take Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page ix.

First Edition (May 1991)

This edition applies to the licensed program IBM Operating System/400 (Program 5738-SS1), Version 2 Release 1 Modification 0, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch serving your locality. Publications are not stocked at the address given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, you may address your comments to:

Attn Department 245
IBM Corporation
3605 Highway 52 N
Rochester, MN 55901-7899

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you or restricting your use of it.

© Copyright International Business Machines Corporation 1991. All rights reserved.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1991

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

AD/Cycle	Application System/400
AS/400	IBM
Operating System/400	OS/400
Personal System/2	RPG/400
SAA	Systems Application Architecture
System/370	400

This publication could contain technical inaccuracies or typographical errors.

This guide may refer to products that are announced but are not yet available.

Information that has changed since Version 1 Release 3 Modification 0 is indicated by a vertical bar (|) to the left of the change.

This guide contains small programs that are furnished by IBM as simple examples to provide an illustration. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. All programs contained herein are provided to you "AS IS". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

Programming Interface

The *GDDM Programming Guide* is intended to help the customer create graphic applications. The guide contains general-use programming interfaces, which allow the customer to write programs that use the services of the graphic data display manager (GDDM).

© Copyright IBM Corp. 1991

About This Guide

This guide, together with the manual *Programming: GDDM Programming Reference*, SC41-0537, describes the application programming interface for the graphics capabilities of the AS/400 system. This guide describes the graphical data display manager (GDDM) and presentation graphics routine (PGR) calls that are available under the AS/400 system. Through the use of sample code and complete programs, along with their associated output, the techniques of graphics programming are demonstrated.

This guide does not describe how to use the IBM AS/400 Business Graphics Utility (BGU), nor does it describe an interface for BGU. For this information, refer to *Business Graphics Utility User's Guide and Reference*, SC09-1408.

You may need to refer to other IBM manuals for more specific information about a particular topic. The *Publications Guide*, GC41-9678, provides information on all the manuals in the AS/400 library.

For a list of publications related to this guide, see the "Bibliography."

Who Should Use This Guide

This guide is to be used by AS/400 application programmers responsible for creating graphics applications.

To use this guide, you should understand the concepts of the IBM Operation System/400 licensed program and the AS/400 system.

You should know how to write, debug, compile, and run programs in one of the following programming languages supported by the AS/400 System: BASIC, the COBOL/400 language, Pascal, PL/I, or the RPG/400 language. Also, you should know how to use one of the supported display devices. You can find information on them in the appropriate PC Support manual.

Throughout this manual, the term *personal computer* applies to any IBM personal computer that uses work station function (WSF), work station emulation (WSE), or 5250 emulation.

© Copyright IBM Corp. 1991

Chapter 1. An Introduction to OS/400 Graphics

What Is OS/400 Graphics All About?

OS/400* (Operation System/400*) Graphics lets you add color and pictures to application programs. You can use the visual effect of color and pictures to help make your programs easier to use and make the results of your high-level language programs easier to understand.

High-level language programs call OS/400 Graphics subroutines (called routines) to construct pictures. Each routine is like a small, self-contained program. The routines are organized in two groups:

GDDM routines

Presentation Graphics routines.

Graphical Data Display Manager (GDDM) routines perform basic graphics tasks, such as drawing a line from point A to point B. A series of these line-drawing routines in an application program can produce a more complex picture. Also, GDDM routines are called in an application program to initialize and end the graphics environment, define characteristics for functions that other GDDM routines will perform (such as setting the color and width of a line that another GDDM routine will draw), send the picture to the work station, and so forth.

Presentation Graphics routines provide you with a fast and efficient way of converting your numeric data to color business charts in an application program. One Presentation Graphics routine can specify the type of chart used to present your data (for example line charts, bar charts, or pie charts). Other routines in your program can do such things as label the data and specify the chart heading.

Presentation Graphics routines are built with sets of GDDM routines. An application program can have any mixture of GDDM and Presentation Graphics routines.

Similar groups of routines are used for graphics on the IBM* System/370* family of data processing systems. If you have access to those systems, you may be able to move your PL/I and BASIC graphics application programs from the AS/400* system.

What Are Some Uses of OS/400 Graphics?

OS/400 Graphics can be used to increase the usability of your existing application programs. For example, a program that uses data description specifications (DDS) display files for menus can be enhanced with GDDM routines. Menu titles and highlighted fields can be shown with larger-than-normal characters in various fonts and colors. Also, graphics pictures, such as company logos, can be added to program menus.

Presentation Graphics routines can be used to change the way the output of a program is presented. A program that produces rows and columns of numbers can be changed so that the program produces that same data graphically on a chart.

Other ideas for uses of OS/400 Graphics are presented in this book.

© Copyright IBM Corp. 1991

What Do I Need Before I Can Use OS/400 Graphics?

AS/400 Hardware

You can write and run OS/400 Graphics application programs using any model of the AS/400 System. (It must have the Operation System/400 program installed.)

Although you can write and compile the programs on any work station that has been described to the system, only the following devices can be used to display graphics:

- IBM personal computer with work station function (WSF)
- IBM personal computer with work station emulation (WSE)
- 5292 Model 2
- IBM personal computer with 5250 emulation

Note: In this manual, the term "graphics work station" means one of those devices.

On the IBM personal computer and IBM Personal System/2* work station, the graphics configuration of the device is determined by the hardware capability and the Virtual Device Interface (VDI) driver loaded in the CONFIG.SYS file. The following table describes the capabilities of each VDI driver:

VDI Driver	Resolution	Colors	Gray Levels
VDIDY004	320 x 200	4	
VDIDY006	640 x 200		2
VDIDY00D	320 x 200	8	4
VDIDY00E	640 x 200	8	4
VDIDY00F	640 x 350		4
VDIDY010	640 x 350	4/8	2/4
VDIDYPGD	640 x 480	8	8
VDIDYA11	640 x 480		2
VDIDYA12	320 x 200	8	8
VDIDY011	640 x 480		2
VDIDY012	640 x 480	8	8
VDIDY013	320 x 200	8	8
VDIDYAF1	1024 x 768	8	8
VDIDYAF2	1024 x 768	8	8

When using an IBM personal computer or IBM Personal System/2 work station with 5250 emulation, the following program should be run from your Disk Operating System (DOS) session before attempting to use GDDM graphics on the AS/400 system:

```
> GR5250
```

If this program is not run until after a GDDM application has been started, a CPF8619 error may be generated by GDDM.

These plotters can be attached to graphics work stations:

- IBM 6180 Plotter
- IBM 6182 Plotter
- IBM 6184 Plotter
- IBM 6185 Plotter
- IBM 6186-1 Plotter
- IBM 6186-2 Plotter
- IBM 7371 Plotter
- IBM 7372 Plotter

Graphics can be printed on these Systems Network Architecture (SNA) character string (SCS) devices:

- IBM 4214 Printer
- IBM 4234-2 Printer
- IBM 5224 Printer
- IBM 5225 Printer

Graphics can also be printed on any intelligent printer data stream (IPDS) device that supports graphics, including the following:

- IBM 3812 Printer
- IBM 3816 Printer
- IBM 4028 Printer
- IBM 4224 Printer

It is also possible to send a graphics data format (GDF) file (the internal data GDDM interprets to draw the picture) to other systems. The device receiving the graphics data must have the software necessary to interpret the data.

AS/400 Software

Besides having the OS/400 program installed, you must have a compiler for one of the following high-level languages:

BASIC IBM AS/400 BASIC licensed program product, program number 5738-BA1

IBM SAA* AD/Cycle RPG/400*
IBM RPG/400 licensed program product, program number 5738-RG1

IBM SAA AD/Cycle COBOL/400*
IBM AS/400 COBOL licensed program product, program number 5738-CB1

PL/I IBM AS/400 PL/I licensed program product, program number 5738-PL1 (with library QGDDM in your library list)

Pascal IBM AS/400 Pascal licensed program product, program number 5738-PS1 (with library QGDDM in your library list)

Required Knowledge

To write graphics application programs for the AS/400 System, you must know AS/400 application programming in one of the five high-level languages.

You can learn the concepts and fundamentals of OS/400 Graphics from this manual. Once you have read this manual, you can find more detailed information about GDDM and Presentation Graphics routines in the *GDDM Programming Reference* manual.

In the next chapter, you will learn more about OS/400 Graphics and will see two simple programs, one that shows GDDM routines and one that shows Presentation Graphics routines.

Chapter 2. The Application Program Interface to Graphics

This chapter shows you how GDDM and Presentation Graphics routines can be called by programs to produce graphics (pictures and charts).

In Chapter 1, "An Introduction to OS/400 Graphics," you learned that OS/400 Graphics has two major components, GDDM routines and Presentation Graphics routines. You learned that you can draw pictures by calling the appropriate routines from your high-level language program.

This chapter gives you a brief overview of the steps needed to produce a program, followed by sample programs that show GDDM and Presentation Graphics routines in use.

Note: In the text, the term *GDDM program* means a graphics application program that uses GDDM routines; likewise, a *Presentation Graphics program* is a graphics application program that uses Presentation Graphics routines.

Programming Considerations

To produce an OS/400 Graphics application program on the AS/400 System, you must write an AS/400 application program in a high-level language. You must:

1. Enter the program (the source code).
2. Compile the program (unless you use the BASIC interpreter).
3. Run the program.

You can enter your OS/400 Graphics programs using the Source Entry Utility (SEU). SEU provides special display formats for each of the AS/400 high-level languages to help you enter your programs.

For more information on entering programs with SEU, refer to the *SEU User's Guide and Reference.*

You can compile and run your graphics program in the manner provided by the high-level language you are using. Refer to the appropriate high-level language manual for more information on compiling and running programs.

> Graphics programs written in COBOL/400, RPG/400, Pascal, and PL/I programming languages must be compiled before they can be called and run. BASIC programs can be compiled and run also, but with BASIC you have the option to use the interpreter to LOAD and RUN source files. You can then make changes to your program without having to compile it again.
>
> The Command Language (CL) command STRBAS starts the BASIC interpreter. For more information, refer to the *BASIC User's Guide and Reference.*
>
> The early program examples in this manual are all written in BASIC. Chapter 6, "Graphics Application Program Examples," shows programs written in all the supported languages.

© Copyright IBM Corp. 1991

Drawing a Simple Picture with GDDM

Many of the things you do to draw a picture with GDDM are similar to things you do to draw a picture with pencils and paper.

When you draw a picture with pencils and paper, you perform these steps:

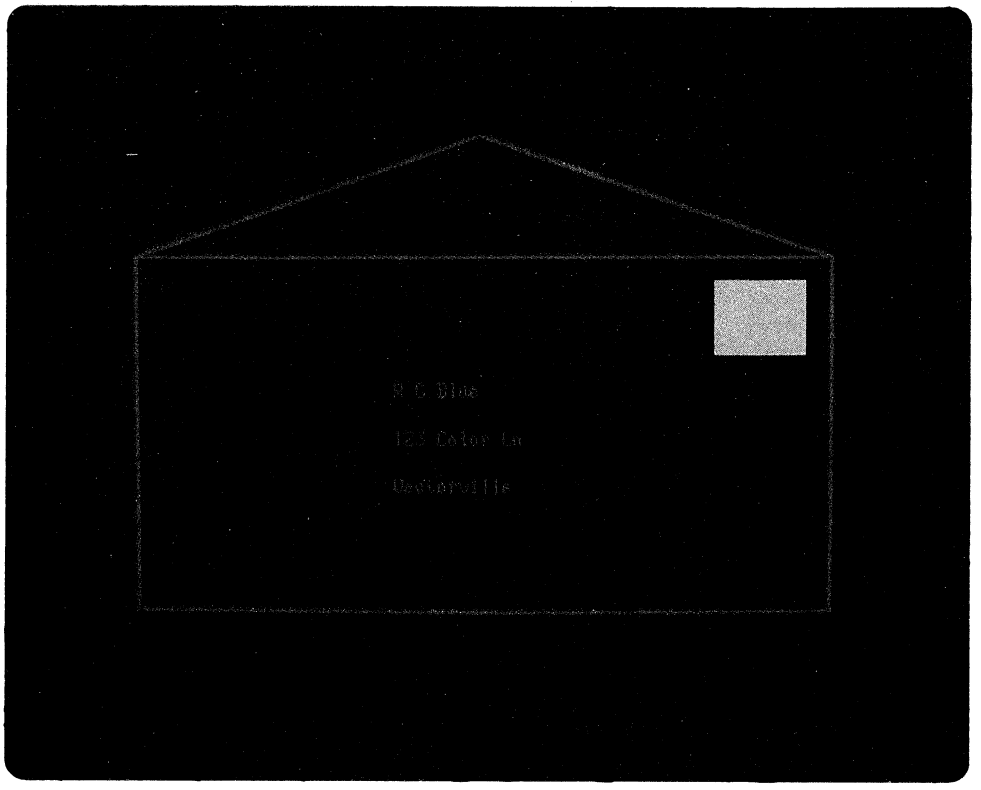
1. *Initialize the graphics environment.* You decide to draw a picture.
2. *Set the initial attributes.* You select a pencil of the type and color you want to use for your picture.
3. *Draw the picture.* You place the tip of the pencil at the point on a paper where the first line should begin. Then you draw the line to another point on the paper. If you decide the next line should be of a different color, you get the appropriate colored pencil. You position the pencil on the paper (perhaps on the end point of the first line), then draw the next line. You continue to draw lines until the picture is complete.
4. *Display the picture.* You look at it or give it to someone else to look at.
5. *End the graphics environment.* You decide the picture is complete.

A graphics program on the AS/400 System uses GDDM routines to perform the same basic steps:

1. *Initialize the graphics environment.* The program uses a GDDM routine to signal to the AS/400 System that it is going to call GDDM or call both GDDM and the Presentation Graphics routines.
2. *Set the initial attributes.* The attributes (visual characteristics) of the first item to be drawn are defined by the appropriate GDDM routine. For example, if the first item to be drawn is a line, the program must use GDDM routines to specify the color and line width attributes (the color and type of the pencil). Any other lines that the program draws after the first line will be of the same color and line width, unless the program sets the attributes to different values for those lines (a different color or type of pencil).
3. *Draw the picture.* The program uses a GDDM routine to specify a point on the screen from which the first line will be drawn (placing the pencil on the paper). Another GDDM routine will then specify the line to draw and the point on the screen where the line is to end (drawing the line to an end point). More GDDM routines can specify more lines that begin at the point each previous line ended, until the picture is complete.
4. *Display the picture.* Once all parts of the picture have been described by the program, another GDDM routine sends the picture to the work station screen or hard-copy device.
5. *End the graphics environment.* After the picture has been sent to the device, a GDDM routine is used to end the graphics environment that was initialized earlier.

Programs that use GDDM and Presentation Graphics routines can be written in the BASIC, COBOL/400, Pascal, PL/I, and RPG/400 high-level languages. BASIC, COBOL/400, and RPG/400 programming languages feature a language extension, the CALL GDDM statement. CALL GDDM is used to call each GDDM and Presentation Graphics routine and to convert data types from those used by the high-level language to those that can be used by GDDM. After the GDDM graphics routine has been performed, the data types are converted back to those used in the high-level language program.

Pascal and PL/I use a slightly different method to call graphics routines. They use *include statements* that can be named in the program. These includes declare the GDDM and Presentation Graphics routines and perform data-type checking.



The envelope. The envelope shown here was drawn by a BASIC program using GDDM routines.

The following BASIC program was used to draw the picture of the envelope:

```
00010 REM ************************************************************
00020 REM                        INITIALIZE
00030 REM ************************************************************
00040 CALL GDDM ('FSINIT')       ! Initialize the graphics environment
00050 INTEGER ATTYPE, ATTVAL, COUNT     ! Declare integer variables
00060 REM ************************************************************
00070 REM                      SET ATTRIBUTES
00080 REM ************************************************************
00090 CALL GDDM ('GSLW',2)               ! Assign line width (2 = wide)
00100 CALL GDDM ('GSCOL',5)              ! Assign color (5 = turquoise)
00110 REM ************************************************************
00120 REM                     DRAW ENVELOPE
00130 REM ************************************************************
00140 CALL GDDM ('GSMOVE',1.0,75.0)      ! Move to upper left corner
00150 CALL GDDM ('GSLINE',80.0,75.0)     ! Draw across to upper right
00160 CALL GDDM ('GSLINE',80.0,1.0)      ! Draw down to lower right
00170 CALL GDDM ('GSLINE',1.0,1.0)       ! Draw across to lower left
00180 CALL GDDM ('GSLINE',1.0,75.0)      ! Draw up to upper left corner
00190 CALL GDDM ('GSLINE',40.0,100.0)    ! Draw up to point of flap
00200 CALL GDDM ('GSLINE',80.0,75.0)     ! Draw down, over to upper right
00210 REM ************************************************************
00220 REM              RESET ATTRIBUTES & DRAW STAMP
00230 REM ************************************************************
00240 CALL GDDM ('GSCOL',2)              ! Assign color (2 = red)
00250 CALL GDDM ('GSAREA',1)             ! Specify filled area w/outline
00260 CALL GDDM ('GSMOVE',67.0,70.0)     ! Move to upper left corner
00270 CALL GDDM ('GSLINE',77.0,70.0)     ! Draw across to upper right
```

```
00280 CALL GDDM ('GSLINE',77.0,55.0)    ! Draw down to lower right
00290 CALL GDDM ('GSLINE',67.0,55.0)    ! Draw across to lower left
00300 CALL GDDM ('GSLINE',67.0,70.0)    ! Draw up to upper left
00310 CALL GDDM ('GSENDA')              ! Fill the area
00320 REM ************************************************************
00330 REM                    WRITE ADDRESS
00340 REM ************************************************************
00350 CALL GDDM ('GSCOL',4)             ! Assign color (4 = green)
00360 CALL GDDM ('GSCHAR',30.0,45.0, 8,'R G Blue')       ! Write line 1
00370 CALL GDDM ('GSCHAR',30.0,35.0,12,'123 Color Ln')   ! Write line 2
00380 CALL GDDM ('GSCHAR',30.0,25.0,11,'Vectorville')    ! Write line 3
00390 REM ************************************************************
00400 REM                  DISPLAY THE PICTURE
00410 REM ************************************************************
00420 CALL GDDM ('ASREAD',ATTYPE,ATTVAL,COUNT)   ! Display the picture
00430 REM ************************************************************
00440 REM                     END GRAPHICS
00450 REM ************************************************************
00460 CALL GDDM ('FSTERM')          ! End graphics
00470 END                           ! End BASIC program
```

As you can see from the comment lines, the program performs these steps:

1. Initialize graphics and declare data types for the variables (all integers, in this instance).
2. Set attributes by assigning values directly to the GDDM attribute routines.
3. Draw the picture by calling the appropriate GDDM routines.
4. Display the picture.
5. End graphics.

Note: The example program is a BASIC program. When BASIC is used for GDDM programs, it has these conventions:

- Integer variables must be declared with an INTEGER statement; integer values can be passed to GDDM routines as literals (the number itself), or they can be passed as variables (variable = the number).
- Floating-point variables need not be declared. Floating-point values can be passed to GDDM routines as literals, or they can be passed as variables. Floating-point values passed as literals must contain a decimal point and at least one digit after the decimal point.
- Character variables with a length less than 19 need not be declared. Character strings can be passed to GDDM routines as literals (the strings must be enclosed in apostrophes) or they can be passed as variables. A character variable assigned a string of 19 or more characters must be declared with a dimension statement (for example, DIM A$*20 sets the dimension of character variable A$ to 20, allowing A$ to be assigned a character string of up to 20 characters).
- GDDM routine names are specified as character strings (enclosed in apostrophes), followed by the parameters of the routine. The syntax for GDDM in COBOL/400, RPG/400, Pascal, and PL/I programming languages differs from that of BASIC. The examples in Chapter 6, "Graphics Application Program Examples," show this same envelope program in the other languages. For more information, refer to the *GDDM Programming Reference* manual.

How the Program Works

Initializing the Graphics Environment: The FSINIT routine initializes the graphics environment.

```
00040 CALL GDDM ('FSINIT')
00050 INTEGER ATTYPE, ATTVAL, COUNT
```

FSINIT signals to the application program that calls to GDDM, or Presentation Graphics routines, or both will occur.

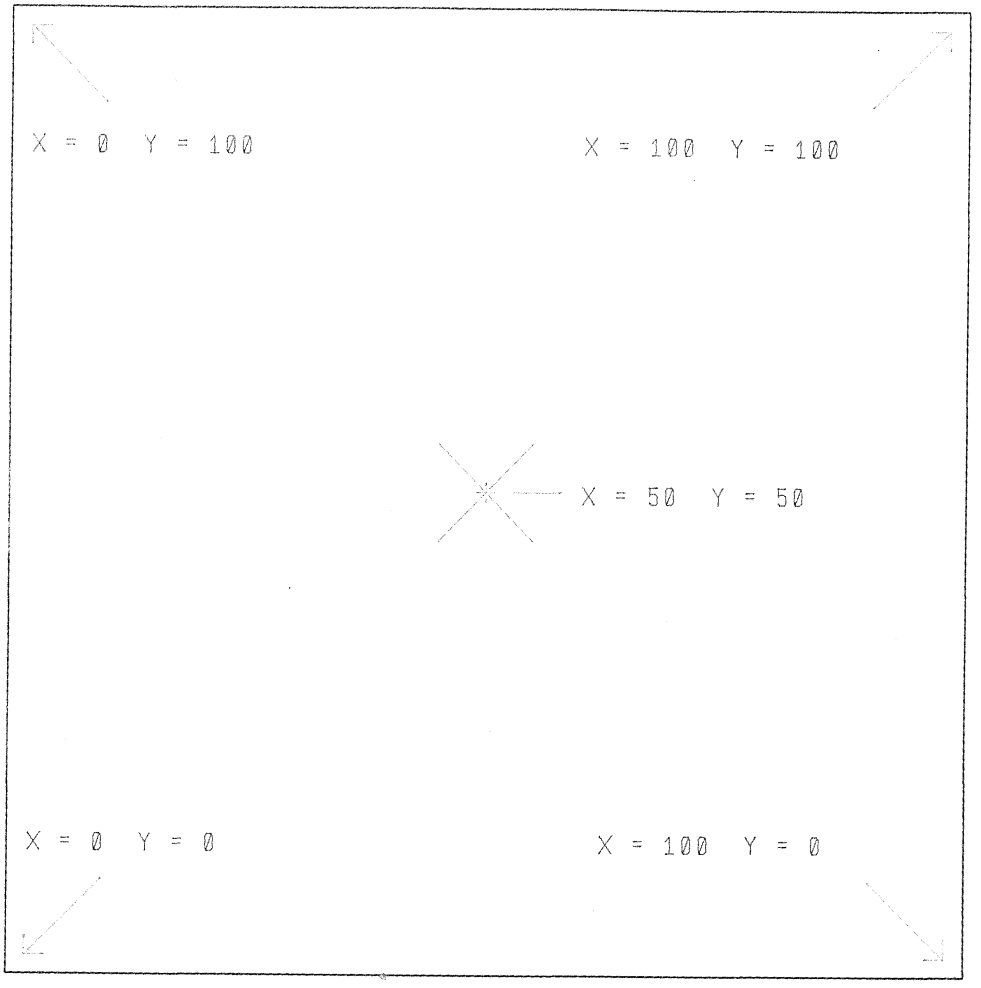
In this program, FSINIT was followed by some data declarations. When you use variables in your program that will be set to integer values, the data types for those variables must be declared. Data type declarations can precede or follow FSINIT.

Setting the Initial Attributes: Attributes determine the results of certain GDDM routines. The attributes that are set in this portion of the sample program define the color and the line width of the envelope outline.

```
00090 CALL GDDM ('GSLW',2)
00100 CALL GDDM ('GSCOL',5)
```

GDDM routines continue to use attribute values until different values are assigned. Default attributes are assigned if no others are specified (line width = narrow, color = green). Other types of attributes were assigned by default in the program. They are discussed in Chapter 3, "Using GDDM."

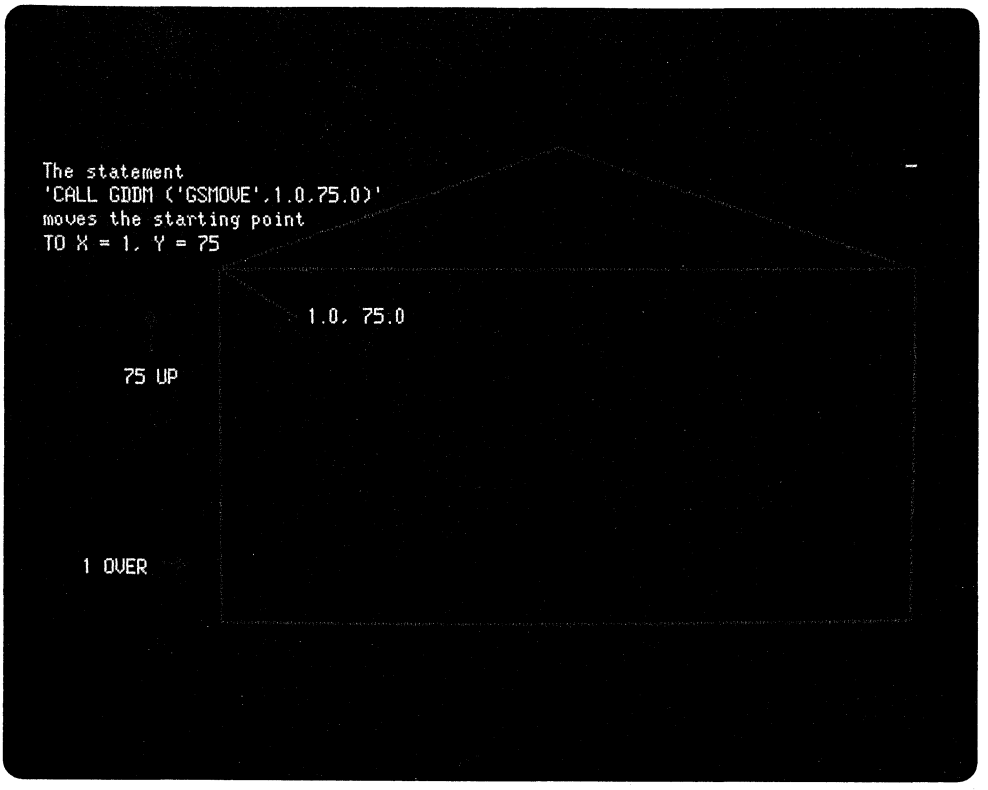
One of the attributes assigned *by default* in the envelope program describes the coordinate system used for positioning the lines and characters of the picture. For this program, the attribute values were the default coordinates of x = 0 through 100, and y = 0 through 100; (x0,y0) is the lower-left position on the picture, and (x100,y100) is the upper-right position.



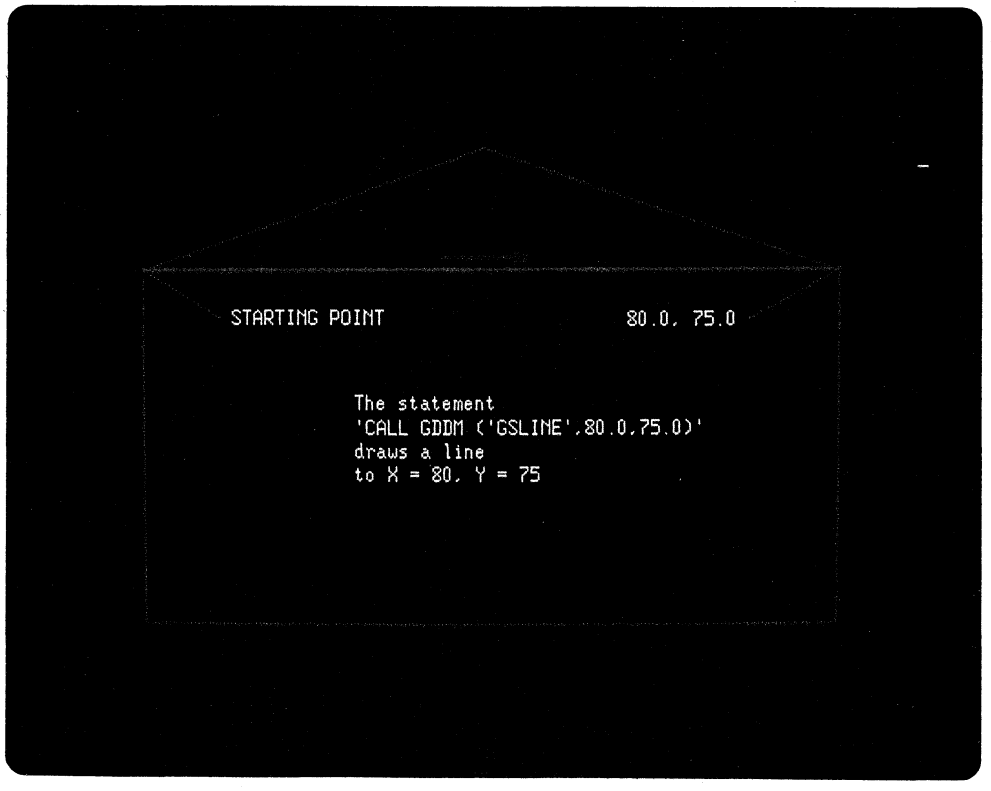
The coordinate system. This illustration shows the default coordinate system used by GDDM when you do not specify one in your program.

Drawing the Picture: GDDM routines define the features of the picture.

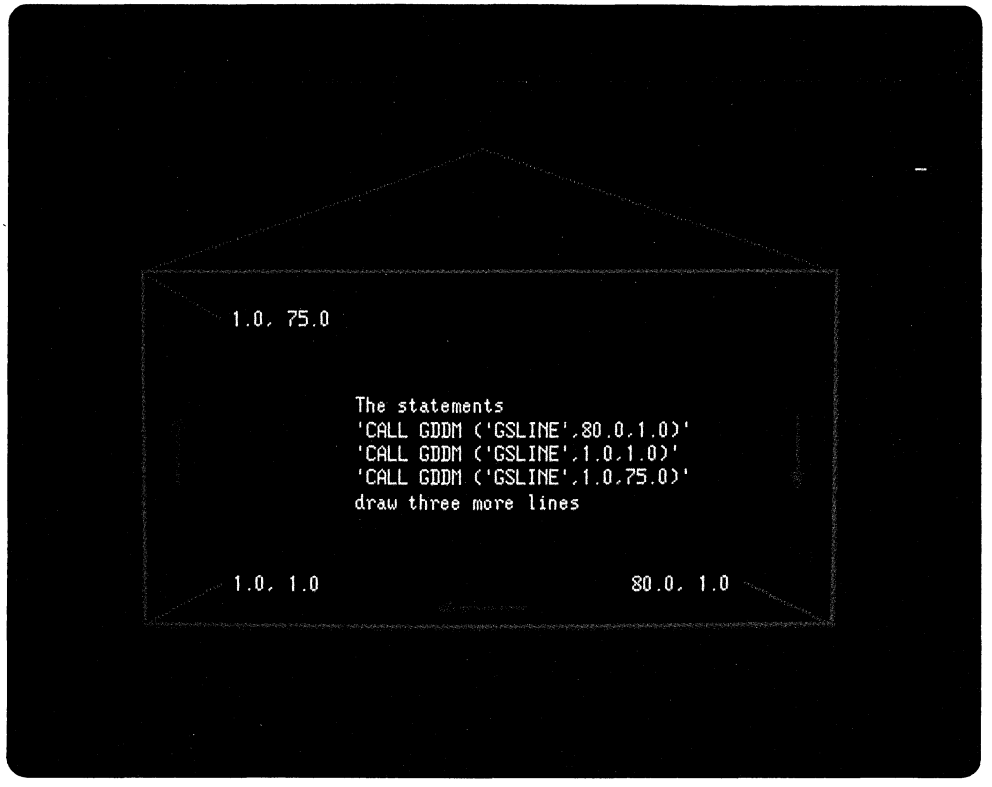
GDDM line-drawing routines use only end point coordinates, so the starting point of the line must be specified. The starting point is called the *current position*. The current position can be the point at which the last drawing routine ended, or you can put the current position wherever you want it with the GSMOVE routine. In the example, GSMOVE places the current position at the upper-left corner of the envelope, coordinates x = 1, y = 75.



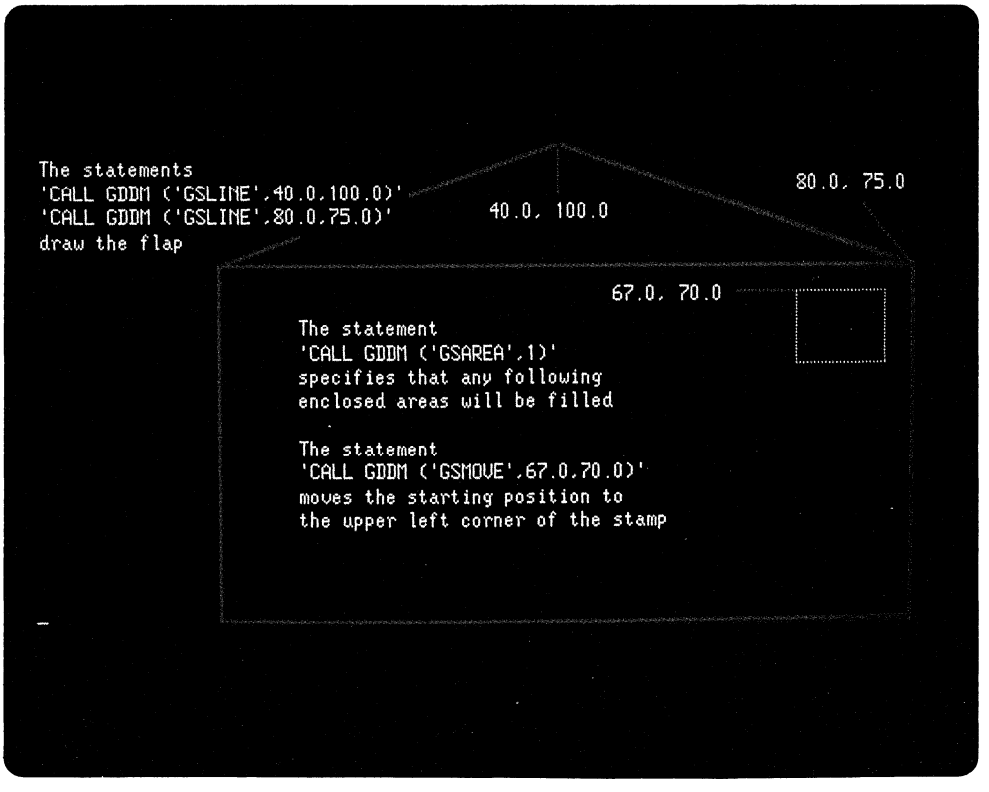
The next routine (GSLINE) draws a line across the top to the upper-right corner of the envelope (the upper-right corner now becomes the current position).



Next, a series of GSLINE routines draw the rectangle part of the envelope outline.



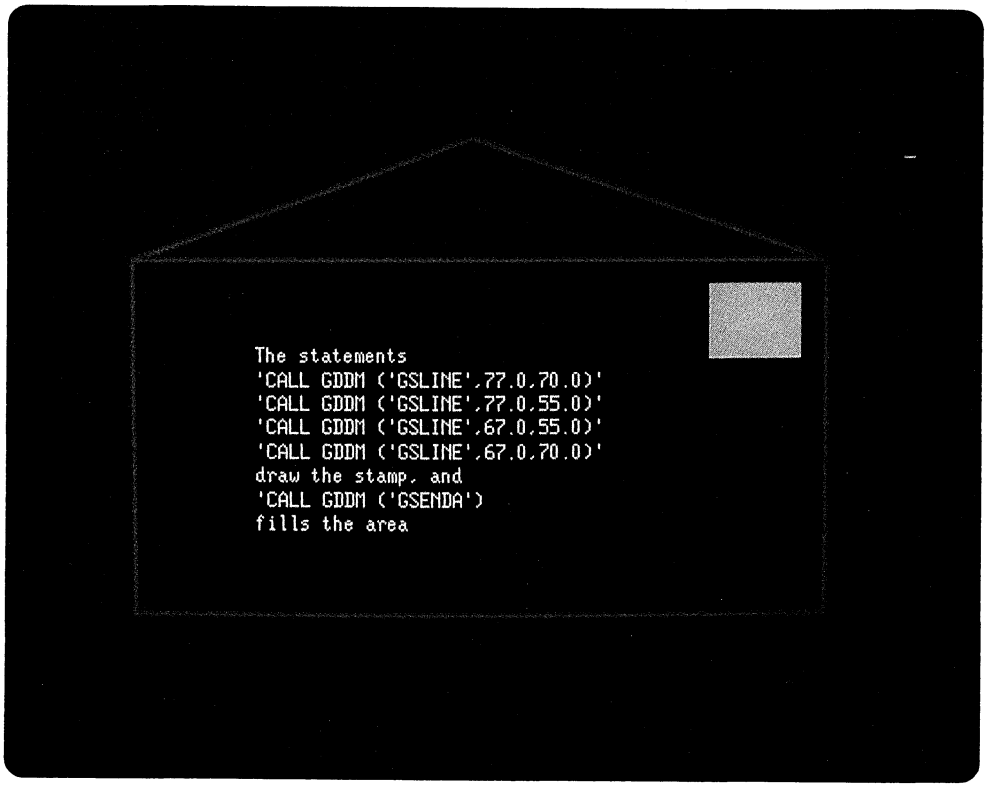
Then, two GSLINE routines draw the envelope flap, which finish the outline of the envelope.



Before the stamp is drawn, the color attribute is changed to 2 (red); otherwise, the stamp would be drawn in the current color (turquoise):

```
00240 CALL GDDM ('GSCOL',2)
```

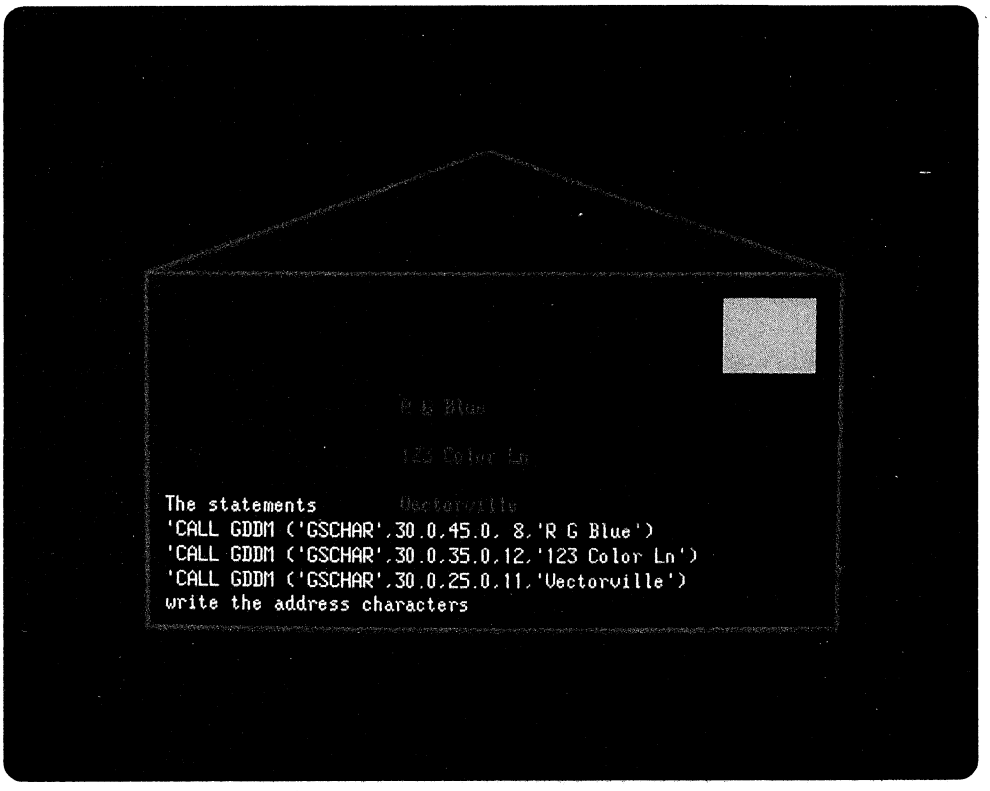
The current position is moved to the starting point for the stamp. Then, the stamp is drawn. By specifying GSAREA before the first GSLINE routine for the stamp and GSENDA after the last GSLINE, the area inside the stamp is filled with a pattern. This is called *area-fill*. You can use a GDDM routine to select a specific pattern for any area-fill, but this area-fill was allowed to default to a solid pattern. The solid pattern is drawn with the new current color (red).



The address is written next. Before routines are called to write the character strings of the address, a new color is assigned (4 — green):

```
00350 CALL GDDM ('GSCOL',4)
```

The characters of the name and address are specified by the GSCHAR routine. GSCHAR specifies the beginning position of the string (the two numbers that follow the routine name), the length of the string, and the string itself (in quotes).



Note that the GSCHAR routine uses all three types of data: floating-point, integer, and character. Floating-point numbers usually specify coordinate values for GDDM routines; the decimal point and following digit must always be included for floating-point numbers used as literal values for GDDM routines in a BASIC program. Integer numbers usually specify a count value or an index value for GDDM routines. Character strings must always be enclosed in single quotes in BASIC programs.

Displaying the Picture: The picture is constructed and held by the system until you ask to see it. Here, the ASREAD routine specifies that the picture should be sent to the work station. It will be shown there until you press a key on the device to acknowledge the picture. Later examples show other uses for ASREAD.

```
00420 CALL GDDM ('ASREAD',ATTYPE,ATTVAL,COUNT)
```

You may have noticed that three integer variables for the ASREAD routine parameters were declared at the beginning of the program, but no values were assigned to those variables in the program.

Parameters like these are assigned values by GDDM; you can later use these program-assigned values for other things in your program. In the case of ASREAD, the ATTYPE variable is assigned a value that corresponds to the type of keyboard response that you give to acknowledge the picture. Your keyboard response could, for example, determine what operations the program performs next: the program will show picture A next if you press a function key, or picture B if you press the Enter key.

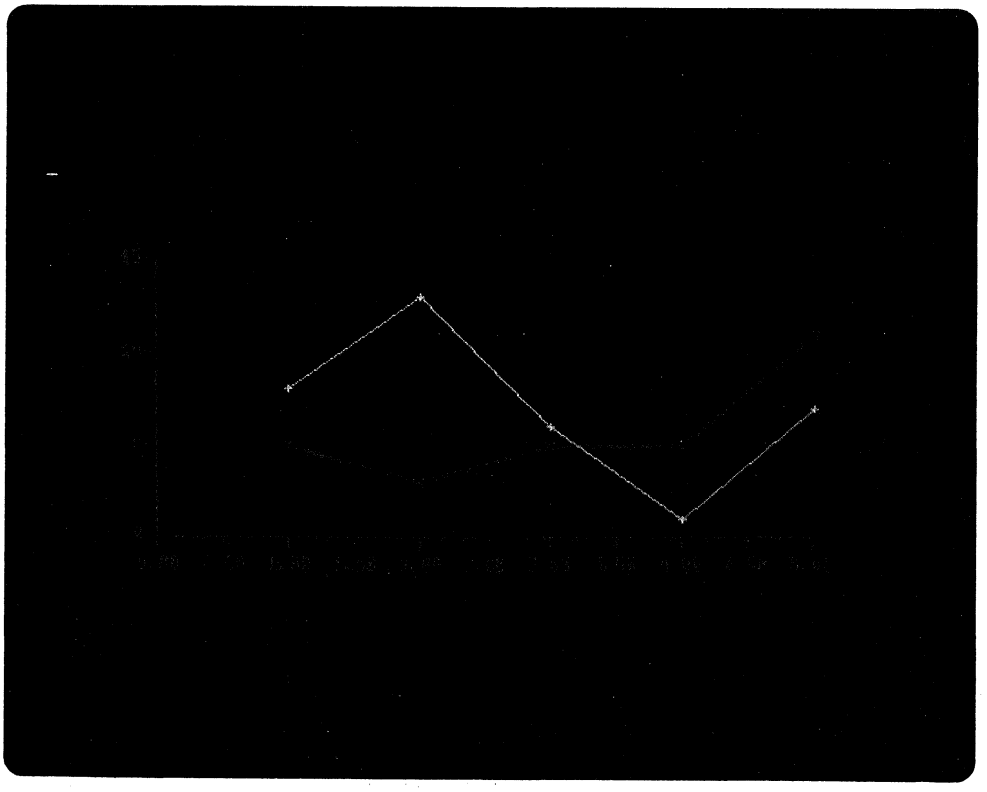
Other GDDM routines also use variables that are assigned values by the program. Many GDDM functions have associated *query* routines that use these program-assigned values. More information about individual query routines can be found in the next chapter.

Ending the Graphics Environment: FSINIT was used at the beginning of the program to initialize the graphics environment. The FSTERM routine is used to end the environment.

```
00460 CALL GDDM ('FSTERM')
00470 END
```

Drawing a Simple Chart with Presentation Graphics Routines

A Presentation Graphics chart-drawing program is basically a GDDM graphics program that calls Presentation Graphics routines used for chart definition. Presentation Graphics routines are IBM-supplied subroutines that contain GDDM routines. For example, a Presentation Graphics routine that draws a line chart can contain several GSMOVE and GSLINE routines that you do not see being called because the Presentation Graphics routine calls the GDDM routines internally.



The line chart. The line chart shown here was drawn by a BASIC program using GDDM and Presentation Graphics routines.

The BASIC program to draw the line chart is:

```
00010 REM ******************************************************
00020 REM                    INITIALIZE
00030 REM ******************************************************
00040 CALL GDDM ('FSINIT')               ! Initialize graphics
00050 INTEGER ATTYPE, ATTVAL, COUNT      ! Declare integer variables
00060 OPTION BASE 1                      ! Set array subscript base
00070 DIM AX(5)                          ! Set array dimensions
00080 MAT READ AX                        ! Read x-array data
00090 DATA 1, 2, 3, 4, 5
00100 DIM AY(10)                         ! Set array dimensions
00110 MAT READ AY                        ! Read y-array data
00120 DATA 5, 3, 5, 5, 11
00130 DATA 8, 13, 6, 1, 7
00140 REM ******************************************************
00150 REM                    DRAW CHART
00160 REM ******************************************************
00170 CALL GDDM ('CHPLOT',2,5,AX(),AY())
00180 ! Format chart with 2 lines, 5 data points per line,
00190 ! with data read from arrays AX and AY
00200 REM ******************************************************
00210 REM                  DISPLAY CHART
00220 REM ******************************************************
00230 CALL GDDM ('ASREAD',ATTYPE,ATTVAL,COUNT)  ! Send chart to device
00240 REM ******************************************************
00250 REM                    END GRAPHICS
00260 REM ******************************************************
00270 CALL GDDM ('FSTERM')               ! End graphics
00280 END                                ! End BASIC program
```

Note: In addition to the programming conventions shown for the BASIC envelope program, GDDM or Presentation Graphics programs that use arrays have these conventions:

- The statement OPTION BASE 1 is used to specify the lowest subscript for array dimensions.
- If the array is an integer array, it must be declared by an INTEGER statement.
- The size (number of elements) for each array must be specified with a DIM statement.
- Several methods exist for assigning data to the array. This program uses the MAT READ statement followed by DATA statements that hold the array data to be read.

For information on conventions for BASIC, COBOL/400, Pascal, PL/I, and RPG/400 programs, see the *GDDM Programming Reference* manual, and the examples in Chapter 6, "Graphics Application Program Examples," that show this program in the other languages.

How the Program Works

Initializing the Graphics Environment: A Presentation Graphics program must be initialized like a GDDM one. FSINIT initializes the graphics environment. Data declarations must also occur if integer variables are used by the routines.

The Presentation Graphics line graph uses two arrays (the elements in these arrays are passed as data to the CHPLOT routine):

```
00050 INTEGER ATTYPE, ATTVAL, COUNT
00060 OPTION BASE 1
00070 DIM AX(5)
00080 MAT READ AX
00090 DATA 1, 2, 3, 4, 5
00100 DIM AY(10)
00110 MAT READ AY
00120 DATA 5, 3, 5, 5, 11
00130 DATA 8, 13, 6, 1, 7
```

Drawing the Chart: After initialization, the chart is drawn (plotted) by the CHPLOT routine. CHPLOT uses values that specify the components (the number of lines being charted that represent data), the count (the number of data points in each line that represent data), and the data (in this case, the names of the arrays that contain the data).

```
00170 CALL GDDM ('CHPLOT',2,5,AX(),AY())
```

Reference lines are used on a chart so that you can interpret that data shown.

The vertical and horizontal reference lines on a chart are called axes. Usually the x axis is horizontal and the y axis vertical.

The x axis usually shows the *independent variable*. The independent variable is an array of values that are independent of the y axis values. A common independent variable is time (minutes, days, weeks, months).

The y axis usually shows the *dependent variable*. The dependent variable is an array of values that correspond to the independent variable array.

For example:

If the independent variable is hours of the day (1 through 24), the dependent variable could be a temperature reading taken each hour. So that you could see that at 10:00 the temperature was 67 degrees F.

In this case, the value of the y variable (temperature) depends on the corresponding x variable (hour).

Displaying the Chart: The output from the Presentation Graphics program is sent to the display device with the GDDM routine ASREAD.

```
00230 CALL GDDM ('ASREAD',ATTYPE,ATTVAL,COUNT)
```

Terminating the Graphics Environment: FSTERM terminates the graphics environment as it did in the sample GDDM program.

```
00270 CALL GDDM ('FSTERM')
00280 END
```

The Syntax of Routines

Parameters that Supply Values to the Program

As you can see from the sample programs, both GDDM and Presentation Graphics use *routines* to perform graphics tasks. In many instances, the result of that task depends on the values of the routine's *parameters*. For example, the GDDM routine GSCOL used the integer number 2 for its parameter to set the current color red.

The GSCOL routine has a single parameter:

```
GSCOL(color-code)
```

Some of the routines have *parameter lists*. The GSCHAR routine has a parameter list:

```
GSCHAR(x,y,length,string)
```

The parameters in routines are always of a specific *data type*. Data types can be:

4-byte binary integer

Short floating point

Character string.

Some routines have a mixture of all three data types in their parameter list. In GSCHAR, for example, **x** and **y** are floating-point values, **length** is an integer value, and **string** is a character string.

Note: The manual *GDDM Programming Reference* shows the data types required for each parameter value in the descriptions of the routines.

Parameters that Get Values from the Program

Some routines have parameters that receive values *from* GDDM, rather than passing values *to* GDDM. These values are *returned* by the parameters. You can use the values in returned parameters to do various things in your program. For example, you can use the type of keyboard response returned to the ASREAD routine to control which step of the program is performed next.

Like parameters that pass values to GDDM, parameters that receive values must be declared as variables of the correct data type.

Many routines that receive values from GDDM are *query* routines. Query routines get current values from GDDM. For example, GSQCOL returns the current value of the *color* attribute. You can use similar query routines to get the value of an attribute, or to discover device characteristics or picture dimensions.

The Names of the Routines

GDDM routine names are 4- to 6-character mnemonics whose first two characters are AS (input/output function), FS (program function), GS (graphic function), or DS (device function).

The other characters in the name represent the function of the routine (GSENDA, for example, is a routine that ends an area-fill). When the character Q follows the first two characters, the routine is a query function (GSQCOL, for example, sets a named variable to the value of the current color attribute).

Presentation Graphics routine names have the same syntax as GDDM names, except that the first two characters of Presentation Graphics routine names are always CH (chart), and no query functions are available with Presentation Graphics routines.

The Parts of a Typical Program

You probably saw similarities between the GDDM envelope program and the Presentation Graphics chart program shown earlier. Both of these programs followed the same steps:

1. Initialize graphics (with FSINIT) and declare variables.
2. Set attributes (the Presentation Graphics program did not).
3. Draw the picture (with calls to the appropriate routines).
4. Send the picture to be displayed (with ASREAD, in this case).
5. End graphics (with FSTERM).

The following chapters show you most of the functions of GDDM and Presentation Graphics routines. You should set up a program with the steps shown so that as you learn about the routines you can experiment with them by inserting them into step 3 of the preceding list of steps. (If you use Pascal, PL/I or any of the *graphics symbol sets* described later, you will need the QGDDM library in your library list.)

Summary of This Chapter

GDDM and Presentation Graphics programs can be written in BASIC, COBOL/400, Pascal, PL/I, and RPG/400 programming languages; the sample programs shown in this chapter were written in BASIC. They were designed to show you some of the concepts of GDDM and Presentation Graphics programs, their similarities and differences.

Programs that are much more complex are possible, of course. Here are some ideas for using GDDM and Presentation Graphics routines:

Programs that use data values from database files. Application programs that use Presentation Graphics routines can be written to access database files and produce business charts based on the data in the files. (The simple line chart you saw earlier could have used data the program read from a database file.)

Programs that prompt for information. GDDM programs can be written that use DDS display files to allow responses to menus that feature graphics. The values entered from the display can then be used to update a file or to provide values for a Presentation Graphics chart.

Programs that show many pictures, one after another. Programs can be written that use the type of keyboard response to control which picture is shown next.

Programs that show many pictures at the same time. Presentation Graphics programs can be written that show more than one chart at a time, and GDDM has routines that define the divisions of the screen and the amount of display screen space allocated to the picture. For example, one picture could show four reduced-size charts using the same or different data. Also, you can design a company logo with GDDM and add a reduced or enlarged-size version of it to business charts or other pictures.

Programs that use dynamic variables. The variables in a GDDM or Presentation Graphics program can be defined and their values passed to the GDDM and Presentation Graphics routines. The values of those variables can be altered by user responses from the keyboard or by the values of data in database files. For example, data that indicates profit could be represented on a chart in green while loss values are shown in red. (Values less than *n* are considered loss, so the program would change the color attribute to red.)

You will find more ideas for uses of GDDM and Presentation Graphics application programs in other chapters of this manual.

Note: Because of the power and the extent of the function available with OS/400 Graphics, complex graphics programs may take longer to run than some of your other application programs. When you call one of these programs, the processing time required is evident by the length of time the input inhibited light on the device remains on. For more powerful models of the AS/400 System, graphics processing takes considerably less time.

In the next chapter, you will learn about the GDDM routines. GDDM routines are versatile and varied and you will see how the routines can work for you. (It is possible to skip to Chapter 4, "Using Presentation Graphics," but to gain a complete understanding of OS/400 Graphics, you should read the chapters in sequence.)

Chapter 3. Using GDDM

In Chapter 2, "The Application Program Interface to Graphics," you learned that high-level language application programs can call GDDM routines to perform graphics functions.

You can use GDDM routines in programs (called graphics application programs) to draw pictures. You can add graphics to existing programs to enhance the usability of display screens, or you can write new application programs that make use of the impact and visual appeal of color graphics for such things as management reports and presentations. This chapter and Chapter 6, "Graphics Application Program Examples" show you pictures generated by GDDM and may give you some ideas for uses of graphics application programs.

Some GDDM routines assign attributes that other GDDM routines use when they are called. Other routines perform some sort of specific action, such as drawing a line or controlling some aspect of the graphics environment.

The first part of this chapter looks at the basic elements of pictures you can draw, as well as the routines you use to draw them and the attributes you can assign them (to make them look the way you want them to look). You can use these basic elements as parts of a more complex picture. (The basic elements you use for drawing pictures are called *primitives*.)

The second part of this chapter deals with GDDM control routines. You can use these routines to handle things in the graphics environment such as picture dimensions and the devices you use.

The third part of this chapter describes support for the graphics data format file. The graphics data format (GDF) file is the output of a graphics program that the AS/400 System uses to build the data stream it sends to a device. The device interprets the data stream and generates the picture. GDDM routines enable you to retrieve the graphics data, which you can then save in a database file or send to another system.

The last part of this chapter is a summary of the information presented about GDDM.

The information that follows describes the functions of the GDDM routines. Routines use integer values, floating-point values, or character values for their parameters; values that you can assign in your program. For a more detailed explanation of each routine, refer to the *GDDM Programming Reference* manual, which shows the syntax of each routine, and shows the data types that need to be declared for each routine parameter.

© Copyright IBM Corp. 1991

Drawing Pictures

In Chapter 2, "The Application Program Interface to Graphics," you learned that you can specify the characteristics (such as color) for basic elements (such as lines and characters) of a picture. Those characteristics were called *attributes.* The basic elements of a drawing are called *primitives.*

Graphic Primitives and Their Attributes

Graphic primitives are the basic items of all OS/400 Graphics pictures. They can be:

- Lines
- Area-fills
- Image symbols
- Graphics symbols
- Markers.

Each type of primitive has specific attributes that can be assigned to it. In most instances, these attributes determine the way one type of primitive looks, but have no effect on any other type of primitive. For example, line type and line width attributes that make lines look the way they do have no effect on characters (graphics symbols).

When you specify an attribute for a primitive, every similar primitive is shown with that attribute until you specify a different attribute. This is called the *current mode* of the attribute. For example, if heavy line is the current mode for the line width attribute, every line drawn is heavy until the current mode is changed to light line.

The only attribute that affects *all* types of primitive is the color attribute, which is discussed next. (All other types of attribute are discussed with their associated primitives.)

Setting Color Attributes

There are two types of color attribute you can specify:

- Color selection
- Color mixing

The *color selection* attribute determines the color of everything drawn in the program until another color is selected. The *color mixing* attribute determines what happens when primitives of different colors overlap.

Selecting a Color

You may recall that the colors for different parts of the envelope picture in Chapter 2, "The Application Program Interface to Graphics" were set by calling the GSCOL routine, which used an integer representing a specific color. This integer is the code for the color.

GSCOL uses the color code to select colors in either of the following ways:

Select color using variable:

```
INTEGER COLOR              ! COLOR is the variable name
LET COLOR = 2              ! Color code 2 = red
CALL GDDM ('GSCOL',COLOR)  ! GSCOL uses COLOR to set color to red;
                           ! every graphics primitive will be red
                           ! until color is changed
```

Select color using literal:

```
CALL GDDM ('GSCOL',6)      ! GSCOL uses integer literal 6 to set color
                           ! to yellow; every graphics primitive will
                           ! yellow until color is changed
```

Each color code corresponds to a GDDM *color definition*. Color definitions are held in a *color table*; the color table holds definitions for eight colors, each identified by a color code. Valid color codes range from −2 through 32,767.

The following chart shows how the color codes are associated with GDDM default color definitions, the resulting color on the graphics work station, the recommended pen number on plotters and the resulting color on printers:

Color Attribute Value	GDDM Color Definition	Display Default Color	618x Plotter Pen No.	7372 Plotter Pen No.	7371 Plotter Pen No.	5224/5, 3812, 3816, 4028, 4214, 4234-2 Printer	4224 Printer
-2	White	White	No pen	No pen	No pen	Background	Background
-1	Black	Black	7	6	1	Black	Black
0	Default	Green	8	6	2	Black	Black
1	Blue	Blue	1	1	1	Black	Blue
2	Red	Red	2	2	2	Black	Red
3	Magenta	Pink	3	3	1	Black	Magenta
4	Green	Green	4	4	2	Black	Green
5	Turquoise	Turquoise	5	5	1	Black	Cyan
6	Yellow	Yellow	6	6	2	Black	Yellow
7	Neutral	White	7	6	1	Black	Black
8	Background	Background	No pen	No pen	No pen	Background	Background
9	Dark blue	Blue	1	1	1	Black	Blue
10	Orange	Red	2	2	2	Black	Red

Color Attribute Value	GDDM Color Definition	Display Default Color	618x Plotter Pen No.	7372 Plotter Pen No.	7371 Plotter Pen No.	5224/5, 3812, 3816, 4028, 4214, 4234-2 Printer	4224 Printer
11	Purple	Pink	3	3	1	Black	Magenta
12	Dark green	Green	4	4	2	Black	Green
13	Dark turquoise	Turquoise	5	5	1	Black	Cyan
14	Mustard	Yellow	6	6	2	Black	Yellow
15	Gray	White	7	6	1	Black	Black
16	Brown	Blue	8	6	2	Black	Brown
17 – 32,767	Beginning with 17, the color definitions for values 9 through 16 are repeated up to 32,767.						

As an example, when your program uses GSCOL to select color code 6 from the default color table, GDDM uses yellow, which shows up as yellow on the display. The plotter, however, uses whatever color pen is in position 6 on the plotter. With the plotter, you can load the pens in any order so that color code 6 could produce any color.

This default color table is easy to use because the colors are the same each time you initialize graphics. For special applications, however, you can define your own color tables for the graphics work station. With your own color table, you can choose eight colors from the total number of colors that are available for your graphics work station. The colors can be assigned to the color codes in any sequence.

Each picture constructed by a GDDM program and shown on the graphics work station can use a maximum of eight colors; one program can produce many pictures, each using a different set of eight colors, but only eight colors can be used in each picture.

Seven colors out of the eight can be changed using the color table, but the eighth, black, cannot be changed.

You can define colors for your color table by assigning values to the three components of a color: hue, lightness, and saturation (HLS). The default color table for the display contains these definitions for the GDDM default colors:

Color Code	Display Default Color	H (Hue)	L (Lightness)	S (Saturation)
1	Blue	0.0	0.5	1.0
2	Red	0.33333	0.5	1.0
3	Pink	0.16666	0.5	1.0
4	Green	0.66666	0.5	1.0
5	Turquoise	0.83333	0.5	1.0
6	Yellow	0.5	0.5	1.0
7	Neutral (white)	0.0	1.0	0.0
8	Background (black)	0.0	0.0	0.0

Hue, lightness, and saturation are assigned with a value less than or equal to 1.0 but greater than or equal to 0.0. As you can see from the default color table, default *hue* values extend from 0.0 (blue) to 0.83 (turquoise); the value 1.0 also produces blue. The values of the secondary colors (yellow, turquoise, and pink) are between those of the primary colors (red, green, and blue); for example, the average of red and green hues produces yellow.

With *lightness*, 0.0 indicates no lightness (the color is invisible, or black on the display) and 1.0 indicates the most lightness (white or the neutral color). Each of the colors defined for the default color table (except black and white) use the intermediate lightness value 0.5.

With *saturation*, 0.0 indicates no saturation and 1.0 indicates maximum saturation. The saturation of a color is a measure of how much the color is diluted by white; a fully saturated color is at its brightest, while a color with no saturation is near gray.

The combination of the three HLS values defines color. The total number of possible colors as a result of HLS combinations depends on the type of graphics work station that you are using.

This table shows the HLS definitions for some other colors that might be useful:

Color	H (Hue)	L (Lightness)	S (Saturation)
Orange	0.4	0.5	1.0
Gray	0.0	0.5	0.0
Brown	0.35	0.2	0.75
Purple	0.16	0.2	1.0
Rose	0.22	0.38	0.7
Dark green	0.66	0.2	1.0
Light green	0.6	0.6	0.8
Dark blue	0.0	0.2	1.0
Light blue	0.9	0.7	1.0

Using color

Color can improve the success and visibility of your output. Use color to associate and relate, or separate and distinguish groups of information. Used wisely, it can aid comprehension and attract attention. How you use color depends on the information you want to communicate.

In pictures and customized menus, a color table that uses subtle shades (ones with reduced lightness or saturation) can be pleasing and easy to use. A color table made from variations in saturation and lightness of the *same* color can give a pleasing result.

The strongest colors (those in the default color table), you should use only for highlighting and attracting attention.

For charts, use strong colors to draw attention to the important data. For example, use red to show losses on a profit and loss chart. Chart text and reference lines should not distract attention from the data illustrated. Use subtle shades for them.

Experiment with the color tables to see what colors you prefer. An example program for defining color tables is shown on page 6-36.

To define the color table and its entries, use the routine:

GSCTD – Set color table definition. GSCTD specifies the color table identifier, the place in the color table where definitions start (up to seven colors can be defined), the number of entries being defined, and arrays of the values for the HLS components. Selecting a color code in your program that has not been defined in a color table results in the default color value.

Here is an example of a color table definition in which color codes 3, 4 and 5 (usually pink, green, and turquoise) are defined to show brown, purple, and orange. When the color table is in use, entries 0, 1, 2, and 6, 7, and 8 remain the same as the default color table:

```
OPTION BASE 1                ! Set array subscript
      .
INTEGER TABLE, STARTCOLOR, COUNT
      .                      ! Declare integer variables
      .
LET TABLE = 65               ! Color table # 65 (1-64 invalid)
LET STARTCOLOR = 3           ! Start with entry 3
LET COUNT = 3                ! 3 entries will be defined
      .
      .
DIM HUE(3), LIT(3), SAT(3) ! Set up 3 arrays
MAT READ HUE                 ! Read the data that follows
DATA 0.35, 0.16, 0.4         ! Hue component for each color
MAT READ LIT                 ! Read data
DATA 0.2, 0.2, 0.5           ! Lightness components
MAT READ SAT                 ! Read data
DATA 0.75, 1.0, 1.0          ! Saturation components
      .
      .
```

```
CALL GDDM ('GSCTD',TABLE,STARTCOLOR,COUNT,HUE(),LIT(),SAT())
                          ! Define color table
CALL GDDM ('GSCT',TABLE)   ! Select the defined color table
```

Only the first seven entries of the color table can be defined. The eighth entry is reserved for the background color (black) which can be used to *erase* parts of the picture.

Color tables only apply to graphics work stations. For plotters, the colors are determined by the position of the pens; the GDDM colors are only produced on the plot if pens of appropriate colors are loaded into the carousel so that they correspond to the color definitions of the color table.

You can define as many color tables as you want for the device currently in use. To define these tables, you can include the necessary program statements in your program, or you can call from your program another program written specifically for defining color tables. An example of such a program is shown on page 6-36.

You can assign the color code sequence in the color table in any order. One color table can be selected per *page*; a page is one picture or one screen of graphics. Therefore, you can only show a maximum of seven colors at the same time in one picture. For more information, see page 3-44.

Because color tables are defined only for the device currently selected, any defined color tables are lost when you select a different device. For information about GDDM device control routines, see "Device Controls" on page 3-64.

When a color table has been defined, you must make it the current color table by selecting it:

GSCT – Select a color table. GSCT selects a defined color table for use. After a color table is selected, the color code values used in the GSCOL routine correspond to the color entries in the color table.

The following routines are also used for color table functions:

GSQCTD – Query a color table definition. GSQCTD returns the definition of a color table. The values of each of the HLS components are returned to the named arrays. You can use this routine, for example, to query the color table definition, then increment elements of the HLS arrays with a mathematical operation, and use the incremented values to define a new table.

GSQCT – Query the current color table. GSQCT returns the identifier of the currently selected color table.

Your program can find out the current color code value at any time by using the routine:

GSQCOL – Query current color. GSQCOL returns the color code value.

You could use the GSQCOL routine like this:

```
10 LET COLOR = 1                ! Color code 1 = blue
20 CALL GDDM ('GSCOL',COLOR)    ! GSCOL uses COLOR to set color
       .
       .
40 IF SOMETHING <= 20, THEN 60 ELSE 90
50                          ! Test value of SOMETHING, get color
60 CALL GDDM ('GSQCOL',COLOR)  ! Query current color
70 LET COLOR = COLOR+1         ! Increment code by 1
80 CALL GDDM ('GSCOL',COLOR)   ! GSCOL uses COLOR to set color
90 REM
```

Assume that the default color table is the current color table. Line 20 makes blue the current color. Line 40 tests the value of the variable SOMETHING; if SOMETHING is less than 20, line 60 queries the current color (blue − 1), line 70 adds 1 to that value, and line 80 sets the current color to red (1 + 1 = 2, the color code for red).

Mixing Colors

The color mixing attribute controls the way a primitive's color is mixed with the color of another primitive when they intersect.

The following routine is used to set the mode of color mixing:

GSMIX − Set color mixing. GSMIX sets the color mixing mode, using the following values:

0 Default (same as 2)
1 Mix (OR of color codes; not supported on plotters, defaults to 2)
2 Overpaint
3 Overpaint (same as 2)
4 Mix (*exclusive* OR of color codes; not supported on printers, defaults to 2)

By default, the second color drawn *overpaints* the first. For example, if GSMIX is not specified or if GSMIX(0 or 2) is specified, the junction of a red line drawn over a green one is red (the red line overpaints the green).

If GSMIX(1) is specified to set color mixing on and the default color table is in use, the junction of the red and green line is the mixture of red and green, namely yellow. The following table shows the results of color mixing with mode 1 for the *default color table*:

Color	Blue	Red	Pink	Green	Turquoise	Yellow	Neutral
Blue (B)	B	P	P	T	T	N	N
Red (R)	P	R	P	Y	N	Y	N
Pink (P)	P	P	P	N	N	N	N
Green (G)	T	Y	N	G	T	Y	N
Turquoise (T)	T	N	N	T	T	N	N
Yellow (Y)	N	Y	N	Y	N	Y	N
Neutral (N)	N	N	N	N	N	N	N

The plotter is always in a *modified mix* mode, where the color is the result of etching one color over the top of another.

On the graphics work station, a primitive drawn in black will be visible only if color mixing 2 is in effect, and when drawn over another color. For example, drawing in

black on a colored background produces a reverse-video effect. Used with solid shading, it can blank out or erase an area (to erase an area, the entire screen is erased and drawn again).

If GSMIX is specified to set color mixing on and a modified color table is in use, the junction of the colors defined for color table codes 3 and 4 is drawn in the color defined for color code 7, rather than being the result of an additive color process. The result of color mixing for any color table (including the default table) is determined as follows: the binary equivalent of one color index is ORed or exclusively ORed with the other. The result of this operation is the color code of the *mixed* colors.

This table shows, using color codes, the results of mixing for any color table, default or modified:

		Original Color						
		1	**2**	**3**	**4**	**5**	**6**	**7**
	1	1	3	3	5	5	7	7
	2	3	2	3	6	7	6	7
Mixed	**3**	3	3	3	7	7	7	7
With	**4**	5	6	7	4	5	6	7
	5	5	7	7	5	5	7	7
	6	7	6	7	6	7	6	7
	7	7	7	7	7	7	7	7

If GSMIX is specified to set color mixing on with option 4 (exclusive-OR of color table index values), the result of color mixing for any color table (including the default table) is determined as follows: the binary equivalent of one color code is *exclusively* ORed with the other, which is different from the result of the mix mode selected by option 1.

This table shows, using color codes, the results of option 4 mixing for any color table, default or modified:

		Original Color						
		1	**2**	**3**	**4**	**5**	**6**	**7**
	1	8	3	2	5	4	7	6
	2	3	8	1	6	7	4	5
Mixed	**3**	2	1	8	7	6	5	4
With	**4**	5	6	7	8	1	2	3
	5	4	7	6	1	8	3	2
	6	7	4	5	2	3	8	1
	7	6	5	4	3	2	1	8

The following routine returns the current mixing mode:

GSQMIX – Query the current color mixing mode. GSQMIX returns the value that corresponds to the current mixing mode (0 through 4).

Binary logic

All decimal numbers have a binary equivalent. Binary numbers are made up of 0's and 1's. For example, the decimal number 5 in binary is:

```
                0   1   0   1

The positions:  0   1   0   1

Are worth:      8   4   2   1

Add the values: 0 + 4 + 0 + 1 = 5
```

To OR one binary number with another, each place in this first string is compared with the corresponding place in the second. If either place contains a 1, or if both places contain a 1, the result is a 1. For example:

```
                0   1   0   1     (decimal 5)

                or  or  or  or

                0   0   1   1     (decimal 3)

Gives:          0   1   1   1     (decimal 7)
```

For an *exclusive* OR each place in this first string is compared with the corresponding place in the second. If either place contains a 1, the result is a 1. If both places contain a 1, the result is a 0. For example:

```
                0   1   0   1     (decimal 5)

                or  or  or  or

                0   0   1   1     (decimal 3)

Gives:          0   1   1   0     (decimal 6)
```

The Current Position

Before you start using GDDM routines to draw anything, you have to specify a starting point, which is called the *current position*. If you recall the envelope program in Chapter 2, "The Application Program Interface to Graphics" the first GDDM routine that used coordinates was the GSMOVE routine, which was used to specify the starting point for the first line. When a line was drawn from the current position to new coordinates, the end point of the line became the new current position. To move the current position without drawing, GSMOVE was used again. GSMOVE usually has to be the first routine; otherwise the first thing you draw starts at the lower left-hand corner.

Setting the Current Position

You can use the following routines for setting the current position:

GSMOVE – Set the current position. If no current position has been assigned by GSMOVE or another routine, the value used will be the smallest values of x and y.

Any of the drawing routines. Routines that draw primitives always leave the current position at the end point of the primitive.

Querying the Current Position

You can use the following routine for querying the current position:

GSQCP – Query the current position. GSQCP returns the location of the current position, in x and y coordinates. A retrieved current position value can be passed to another GDDM routine. The following is an example:

```
CALL GDDM ('GSQCP',X,Y)
LET X=X+10
CALL GDDM ('GSLINE',X,Y)
```

GSQCP places the x and y coordinate values into variables X and Y. A value of 10 is added to X, then X and Y are used as values for GSLINE. The result is a horizontal line (10 X-units long) positioned at Y.

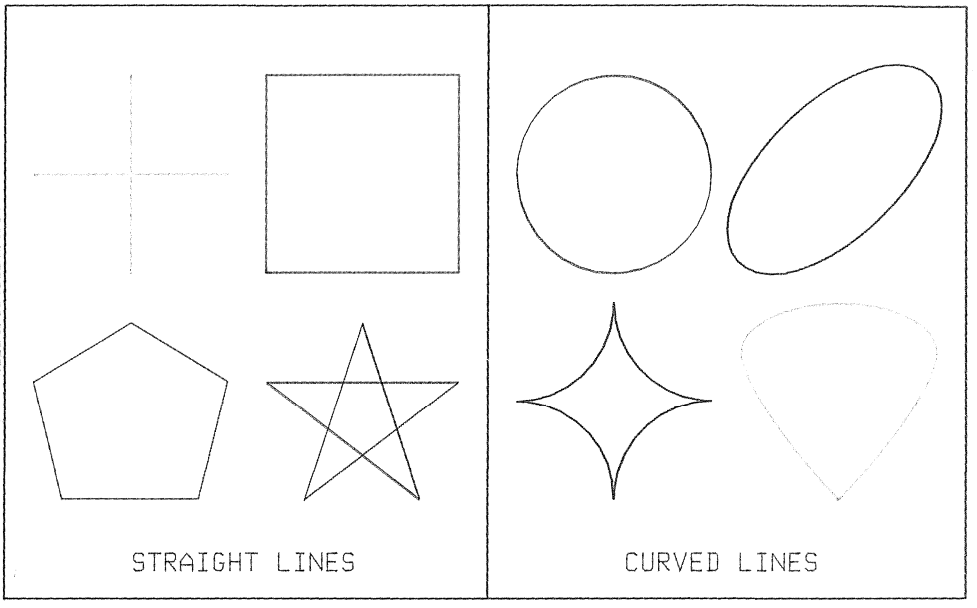
Querying the Cursor Position

GSQCUR – Query the current cursor position. GSQCUR returns the location of the cursor position (in x and y coordinates) at the time of the most recent ASREAD.

An application program can use the values returned by GSQCUR for correlation of the position of the cursor with the x and y position of graphic items on the screen, or for option-selection by cursor position, rather than requiring the user of the program to key in the option name or number.

How to Draw Lines

Lines are the most used primitive in GDDM. Lines drawn by GDDM can be separate, they can cross one another, and they can be connected. Lines can be drawn that are straight or curved. Curved lines can be drawn as arcs, elliptical arcs, or complete circles.



Lines. Your program can use line primitives to draw any shape.

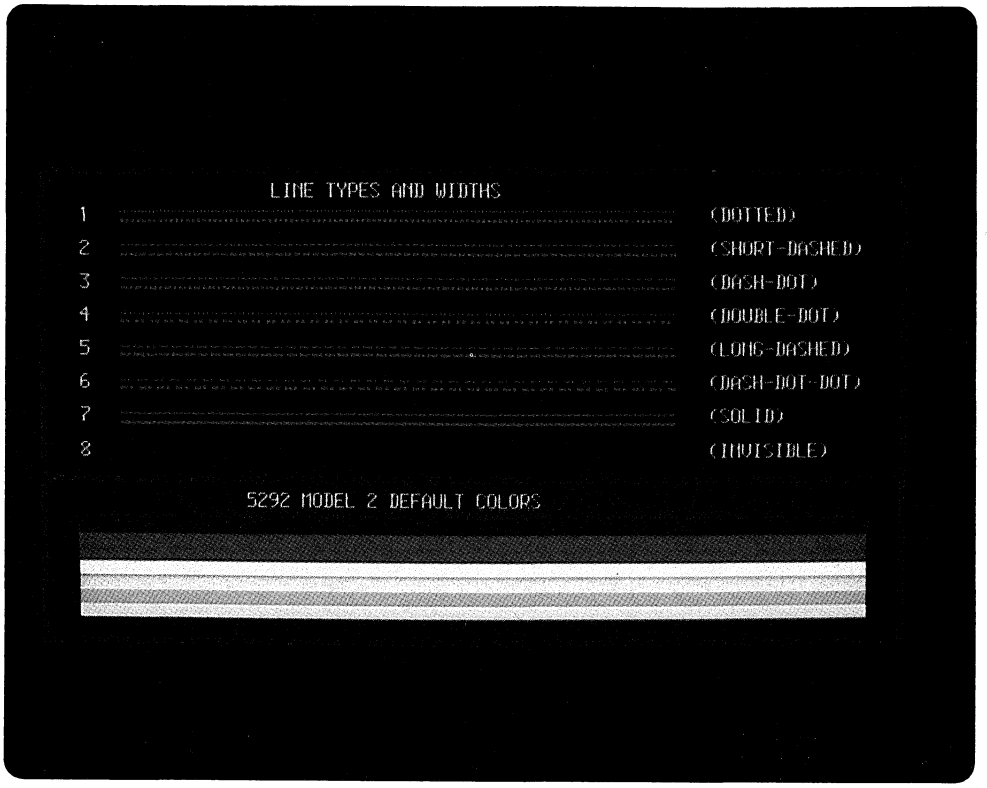
Setting Attributes for Lines

Attributes can be set that assign the type of line and the width of the line.

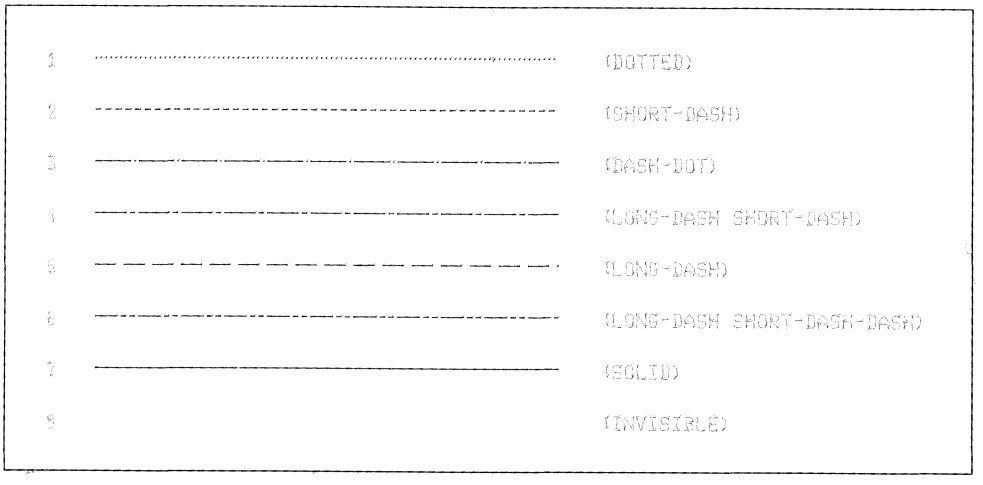
You can use the following routines to set line attributes:

GSLT – Set the current line type. GSLT specifies the type of line to be drawn by GDDM line-drawing functions. If no line type is specified, a solid line is used. GSLT sets the line type by using one of eight line-type codes.

Line attributes. Lines can be drawn in any type and color in either of two widths. The colors shown here can be used for *any* primitive.



Line types for plotter and printer. The line types drawn by the plotter and the printer are different from those shown on the display.



The following routines are also used for line primitives:

GSQLT – Query the current line type. GSQLT returns the value corresponding to the current type of line and sets a variable to that value.

GSLW – Set the current line width. GSLW sets the width of the line to be drawn by GDDM line-drawing functions. There are two line widths: narrow (1) and wide (2). If no line width is specified, a narrow line width is used (GSLW(1)). For the plotter, this routine has no effect; you change the line width by using pens with a larger tip diameter.

Note: Wide lines are drawn slightly longer than narrow lines, so that junctions of wide lines form perfect corners rather than notched corners. Therefore, lines drawn with very little open space between line end points might cause the lines to appear connected.

The IBM PC only supports line width 1.

GSQLW – Query the current line width. GSQLW returns the current line width and sets the variable to that value.

GSFLW – Set the fractional line width. GSFLW specifies the width of the line to be drawn by GDDM line-drawing functions. GSFLW performs the same function as GSLW, except that GSFLW uses a floating-point parameter instead of an integer parameter.

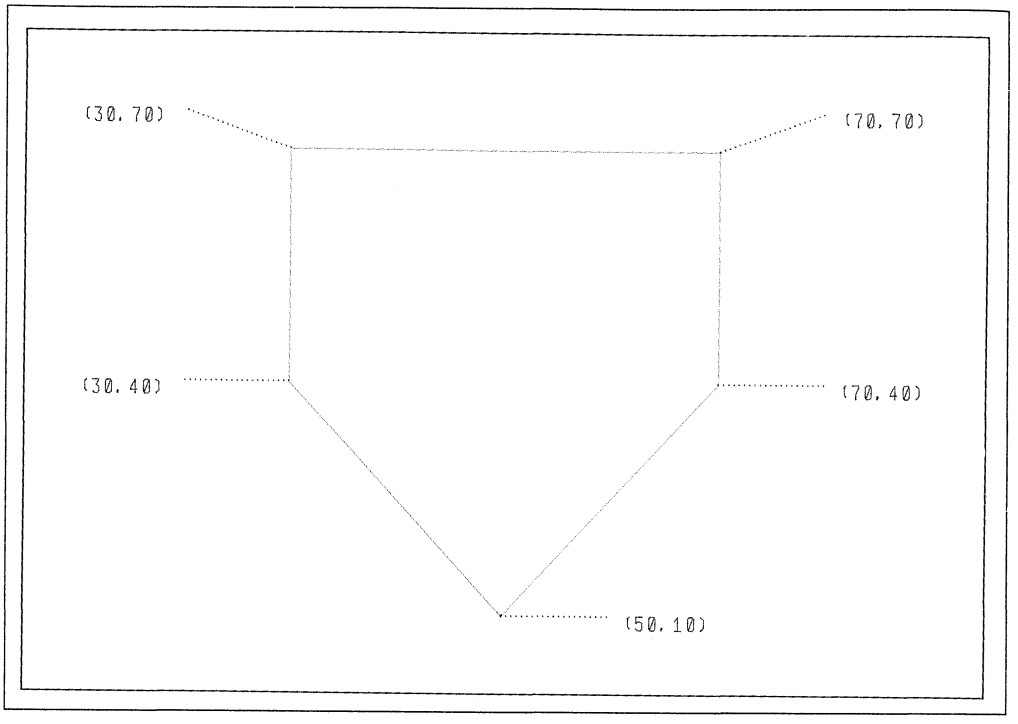
A line width multiplier of 1.0 represents the standard line width for the graphics work station and for the printers. A fractional value of 0.0 through 1.0 draws a narrow line; a value of greater than 1 through 100 draws a wide line.

GSQFLW – Query the fractional line width. GSQFLW returns the current line width as a floating-point value.

Drawing Straight Lines

GSLINE – Draw a straight line. GSLINE draws a single straight line from the current position to the end point specified by x and y.

GSPLNE – Draw a series of lines. GSPLNE draws a series of connected straight lines (a polyline) from the current position. GSPLNE performs many drawing operations with one routine, and can therefore be more efficient than using a series of GSLINE routines.



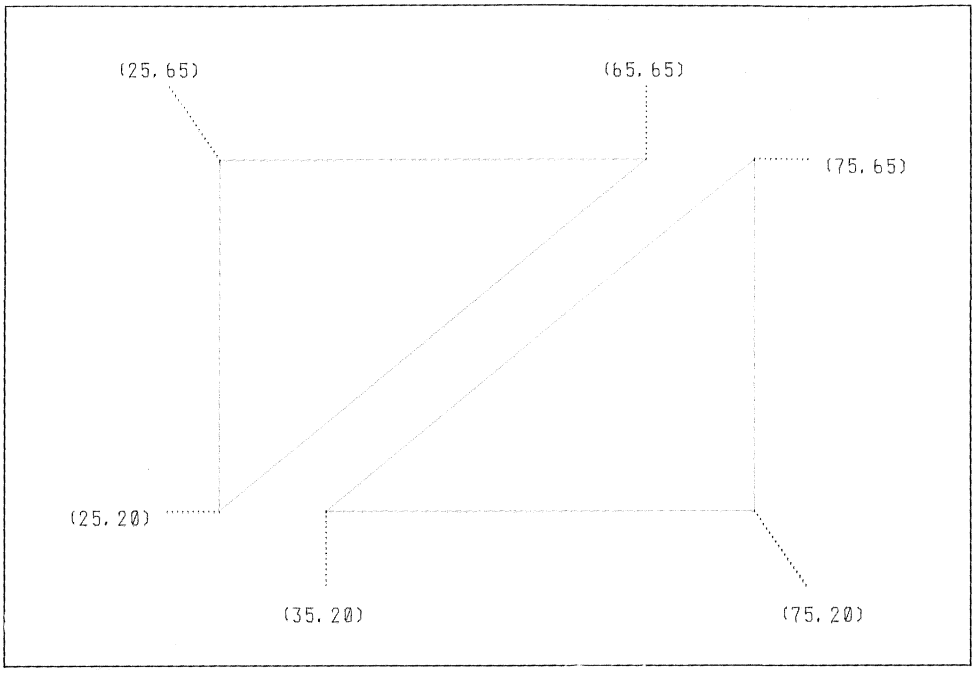
Polyline. A polyline is a series of connected lines.

```
    .
    .
OPTION BASE 1              ! Set array subscript
    .
DIM AX(5) : MAT READ AX    ! Describe and read array
DATA 30,30,70,70,50
DIM AY(5) : MAT READ AY    ! Describe and read array
DATA 40,70,70,40,10
CALL GDDM ('GSMOVE',50.0,10.0)
                           ! Move current position
                           ! to starting point 50,10
CALL GDDM ('GSPLNE',5,AX(),AY())
                           ! Draw 5-part polyline
    .
    .
```

(A *polyline* is a series of connected lines; *polygon* is the geometric term for a polyline that encloses an area.) The end point of each finished line in the polyline becomes the starting point (current position) for the next line.

Attributes cannot be set for individual lines in the polyline drawing. For example, each side of a square polyline cannot be drawn with a different color or line type; for that, you must use a series of GSLINE calls, each preceded by the appropriate attribute setting routine.

GSVECM – Draw a series of vector lines. GSVECM combines the functions of GSMOVE and GSLINE by using an array to specify a series of moves and line drawing functions.



Vector lines. Vector lines are like groups of GSMOVE and GSLINE routines.

```
    .
    .
OPTION BASE 1              ! Set array subscript
    .
DIM VECTOR(24) : MAT READ VECTOR
                           ! Describe and read array
DATA 0,25,20,  1,25,65, 1,65,65, 1,25,20
DATA 0,35,20,  1,75,65, 1,75,20, 1,35,20
                           ! 0 = GSMOVE, 1 = GSLINE
CALL GDDM ('GSVECM',8,VECTOR())
                           ! Draw triangles
```

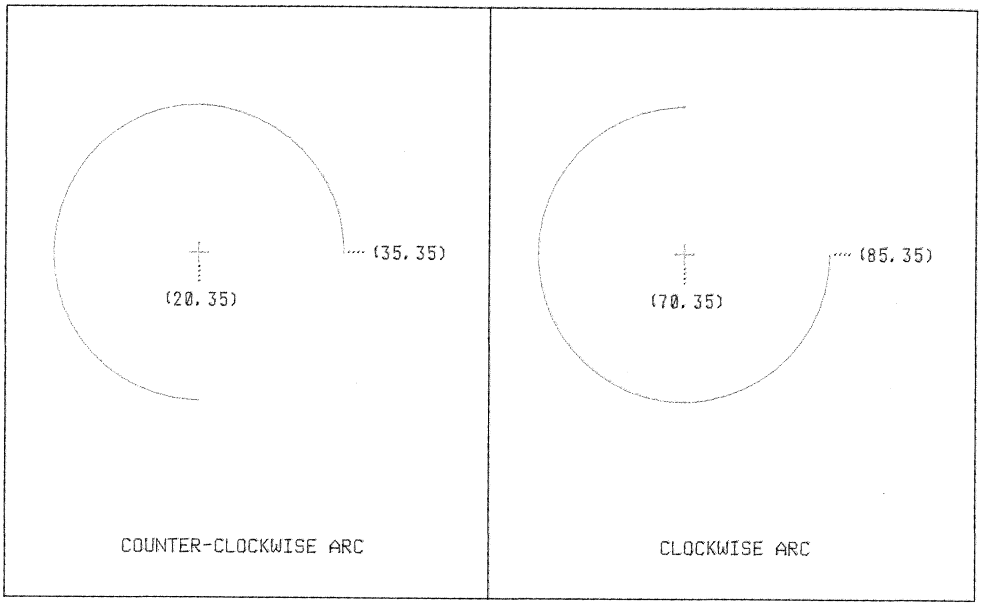
A vector array contains a series of integers in groups of three. The first value of each group specifies a move or a line (0 = move, 1 = line), and the second and third values specify the x and y coordinates for the move or line end point.

Attributes cannot be set for individual lines in the vector drawing.

Drawing Curved Lines

GSARC – Draw a circular arc. GSARC draws an arc around a center point. The arc starts at the current position. The distance the arc sweeps is specified by the angle, and the radius of the arc is determined by the distance between the current position and the specified center point.

Circular arcs. Circular arcs use the current position for the starting point of the arc. The arc routine provides the center point and degree of sweep.




```
! Draw counter-clockwise arc with radius = 15 (35 - 20 = 15)
CALL GDDM ('GSMOVE',35.0,35.0)       ! Move current position
CALL GDDM ('GSARC', 20.0,35.0,270.0) ! Draw arc
    .
! Draw clockwise arc with radius = 15 (85 - 70 = 15)
CALL GDDM ('GSMOVE',85.0,35.0)       ! Move current position
CALL GDDM ('GSARC', 70.0,35.0,-270.0)! Draw arc
    .
```

An angle of 360 degrees results in a full circle; an angle of 90 degrees results in one quarter of a circle. A positive angle provides a counterclockwise sweep; a negative angle provides a clockwise sweep.

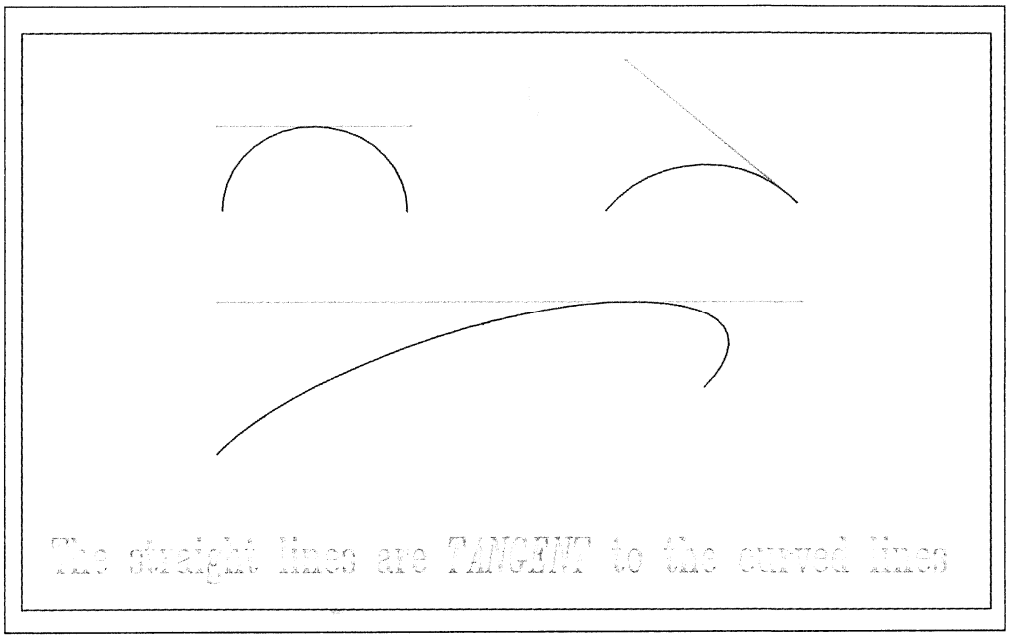
If you cannot easily estimate the degree of sweep and radius needed for an arc to end at a required end point, use GSQCP (Query Current Position) to get the value of the current position after the arc is drawn. If precise placement of the end point of the arc is important, use the GSPFLT routine:

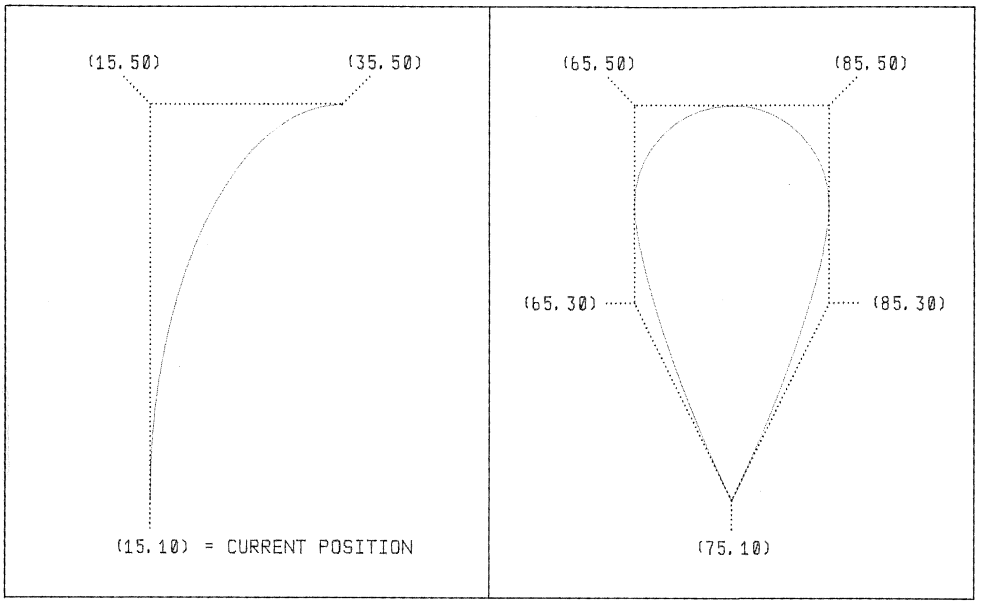
GSPFLT – Draw a series of curved lines. GSPFLT draws a curved line or a series of curved lines.

GSPFLT is similar to the GSPLNE routine, except that GSPFLT uses construction lines that do not appear on the screen as a guide to draw a series of visible, connected, curved lines (called a **polyfillet**).

If only two coordinates are specified by the array, GSPFLT draws two construction lines: one from the current position to the first coordinate, another from there to the second coordinate. A curved line is drawn that begins at the current position and ends at the second coordinate position. The construction lines are tangents to the curved line. The curved line, together with the construction lines, have the appearance of a fillet.

A *tangent* is any straight line that touches a curved line at one point. These are tangents:





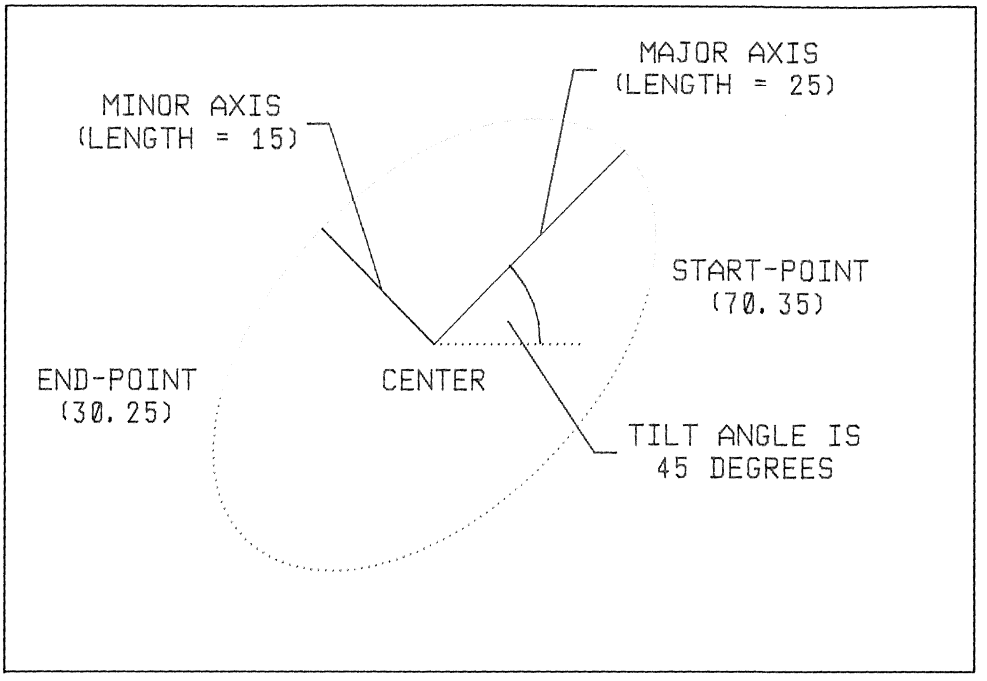
Polyfillets. A polyfillet is an arc or a series of connected arcs built within construction lines.

```
OPTION BASE 1                ! Set array subscript
    .
DIM AX(2) : MAT READ AX      ! Describe and read array
DATA 15,35
DIM AY(2) : MAT READ AY      ! Describe and read array
DATA 50,50
CALL GDDM ('GSMOVE',15.0,10.0)  ! Move to start point
CALL GDDM ('GSPFLT',2,AX(),AY())
                             ! Draw 2-part polyfillet
    .
DIM XX(5) : MAT READ XX      ! Describe and read array
DATA 65,65,85,85,75
DIM YY(5) : MAT READ YY      ! Describe and read array
DATA 30,50,50,30,10
CALL GDDM ('GSMOVE',75.0,10.0)  ! Move to start point
CALL GDDM ('GSPFLT',5,XX(),YY())
                             ! Draw 5-part polyfillet
```

A compound polyfillet uses more than two construction lines. When a compound polyfillet is drawn, it starts at the current position and finishes at the end of the last construction line. On its way round inside the construction lines, the polyfillet touches the midpoint of each line. The result is a smooth combination of curved lines with a sharp end point.

You can draw elliptical arcs with this routine:

GSELPS – Draw an elliptic arc. GSELPS draws an elliptic arc from the current position to a specified end point. You describe the arc by assigning the length of the major and minor radii and the degree of tilt, as well as the end point of the arc.



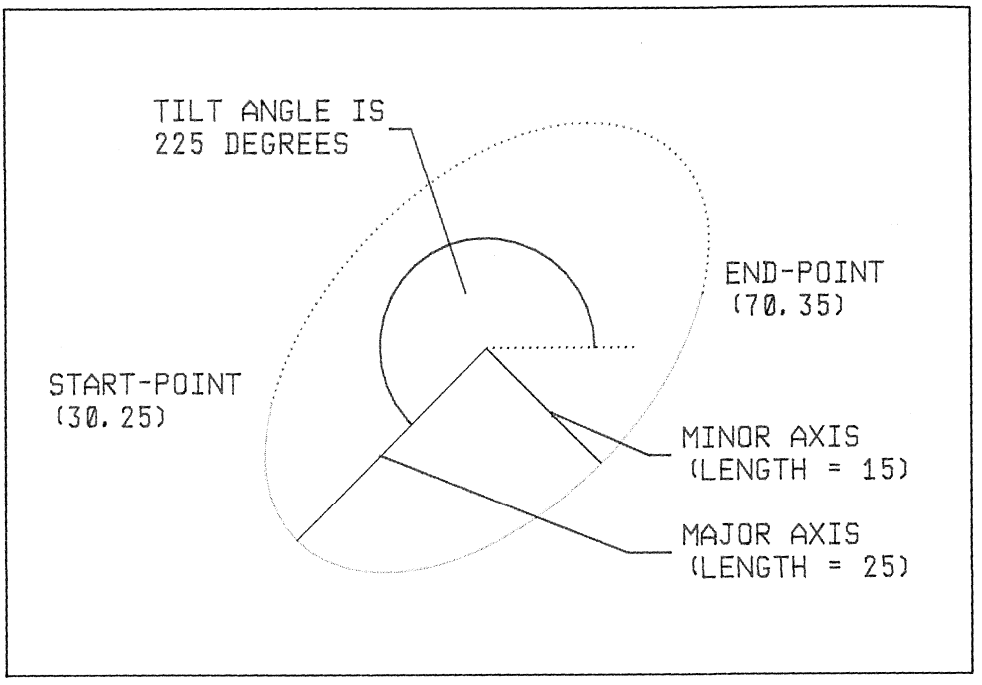
Elliptic arc. An elliptic arc is drawn when the degree of arc and the major and minor radii can accommodate the starting point and end point as points in the arc.

```
CALL GDDM ('GSMOVE',75.0,35.0)  ! Move to starting point
CALL GDDM ('GSELPS',25.0,15.0,45.0,30.0,25.0)
! Draw arc with major radius = 25, minor radius = 15,
! tilt angle 45 degrees counterclockwise, and
! end point x = 30, y = 25
```

An elliptic arc is drawn with these steps:

1. The current position is moved to the starting point of the elliptic arc.
2. An imaginary line the length of the major radius is drawn (1) to the left if the radius value is positive, or (2) to the right if the radius is negative.
3. From the end point of that line an imaginary line the length of the minor radius is drawn (1) up if the radius is positive, or (2) down if the radius is negative.
4. The lines are rotated counterclockwise the number of degrees specified by the positive tiit angle (clockwise by a negative tilt angle).
5. The lines are moved to a position where the arc can hold both the starting point (the current position) and the end point.
6. The elliptic arc is drawn.

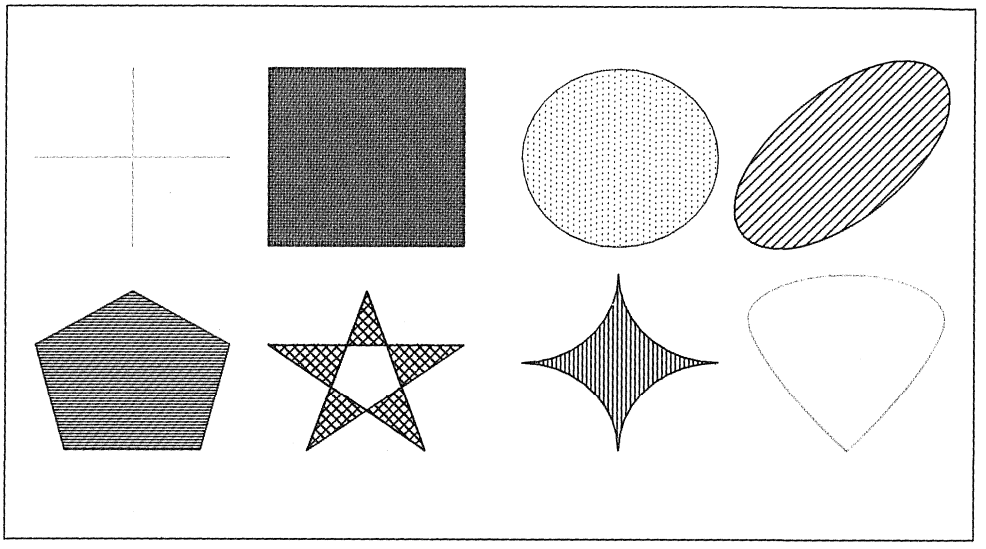
A full ellipse can be drawn with two GSELPS routines. For example, the elliptic arc in the previous illustration could be drawn as a complete ellipse by these routines:




```
CALL GDDM ('GSMOVE',75.0,35.0)  ! Move to starting point
CALL GDDM ('GSELPS',25.0,15.0,45.0,30.0,25.0)
! Draw arc with major radius = 25, minor radius = 15,
! tilt angle 45 degrees counterclockwise, and
! end point x = 30, y = 25
CALL GDDM ('GSELPS',25.0,15.0,225.0,75.0,35.0)
! Draw arc with major radius = 25, minor radius = 15,
! tilt angle 225 degrees counterclockwise, and
! end point x = 75, y = 35 (starting point of first arc)
```

How to Draw Filled Areas

Any of the line-drawing routines can be used to draw an enclosed area, such as a circle or a square. You can use the graphics primitive *area-fill* to shade or color in the enclosed area, using a solid fill or a pattern of the current color.



Filled areas as shown on a plotter. Enclosed areas can be filled with specific patterns.

Drawing Area-Fills

GSAREA – Start a shaded area. GSAREA begins the construction of a shaded area. A parameter specifies whether the outlines of the shaded area are drawn.

GSENDA – End a shaded area. GSENDA ends the construction of a shaded area. When GSENDA is specified, all enclosed areas of the picture (defined since the last GSAREA) are shaded.

Only the following line-drawing or attribute-setting routines can be specified between the GSAREA and the GSENDA routines:

GSMOVE	Set the current position
GSLINE	Draw a straight line
GSPLNE	Draw a series of lines
GSVECM	Draw a series of vector lines
GSLT	Set the current line type
GSLW	Set the current line width
GSCOL	Set the current color
GSMIX	Set the current mix
GSARC	Draw a circular arc
GSPFLT	Draw a series of curved lines
GSCT	Set a color table
GSCTD	Set the color table definition
GSELPS	Draw an elliptic arc.

If you specify a series of lines followed by GSENDA, and the end of the last line is not the same as the starting point of the first line, GDDM adds the last line to enclose the area. This last line is called the *closure line*. The current position then becomes the starting point of the first line.

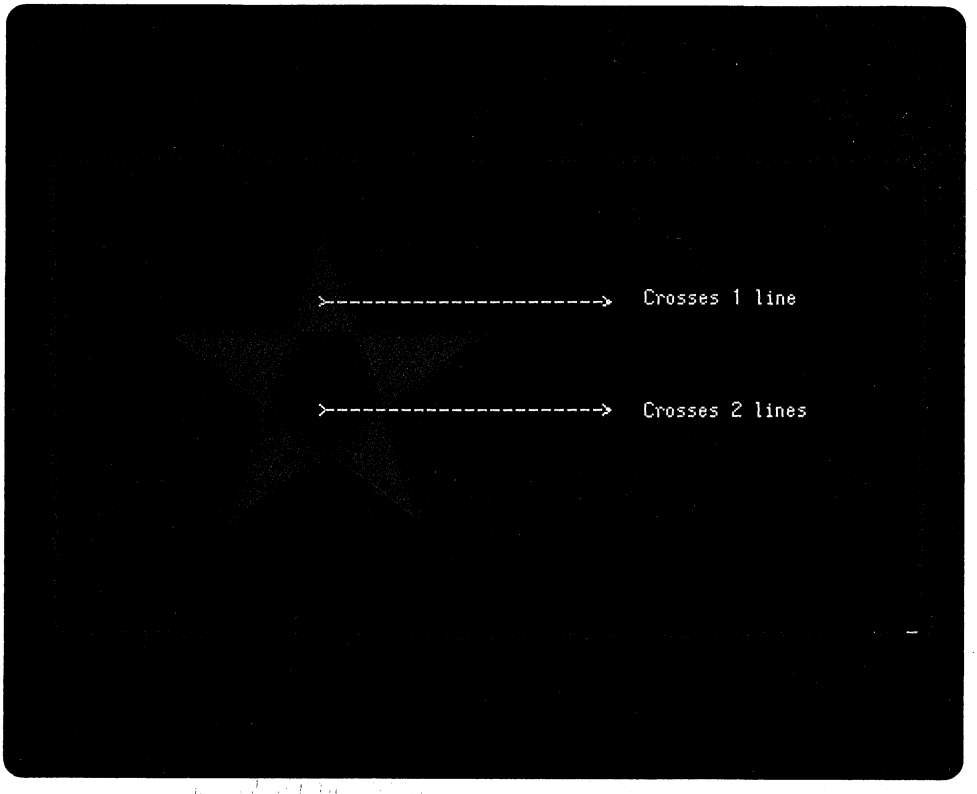
If you want to draw a figure with an outline shown in one color and its shading shown in another, you must:

1. Set the current color (area-fill color).
2. Specify GSAREA with the *outline* option.
3. Change the current color (outline color).
4. Draw the figure.
5. Specify GSENDA.

An area is shaded if two conditions are met:

1. GSAREA has been specified.
2. An odd number of lines must be crossed to leave the enclosed area.

Shading algorithm. An area is shaded when an imaginary line drawn from inside a region crosses an odd number of lines.



The star was drawn with five straight lines that span from one star point to another, rather than 10 lines that form the perimeter of the star.

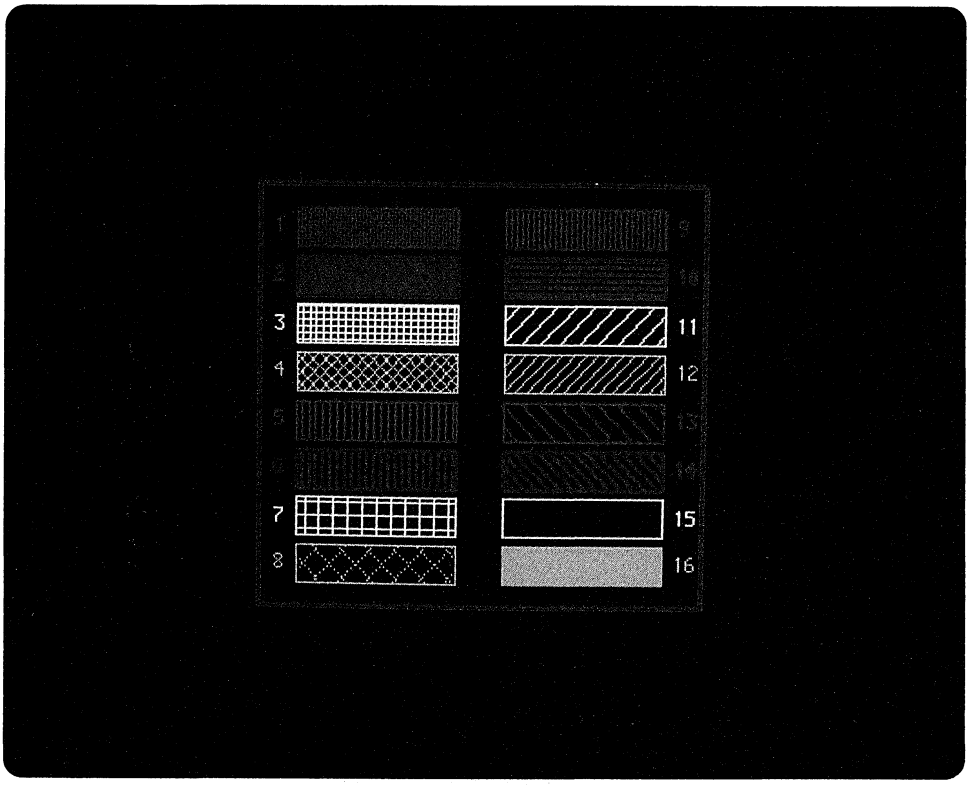
The area inside the star is not shaded because, if you start in that area and move outside the star, you cross two lines. If you start inside one of the shaded star points, you cross either one or three lines to reach the outside; therefore, the area satisfies the odd number of lines requirement and is shaded.

Selecting a Pattern for Filled Areas

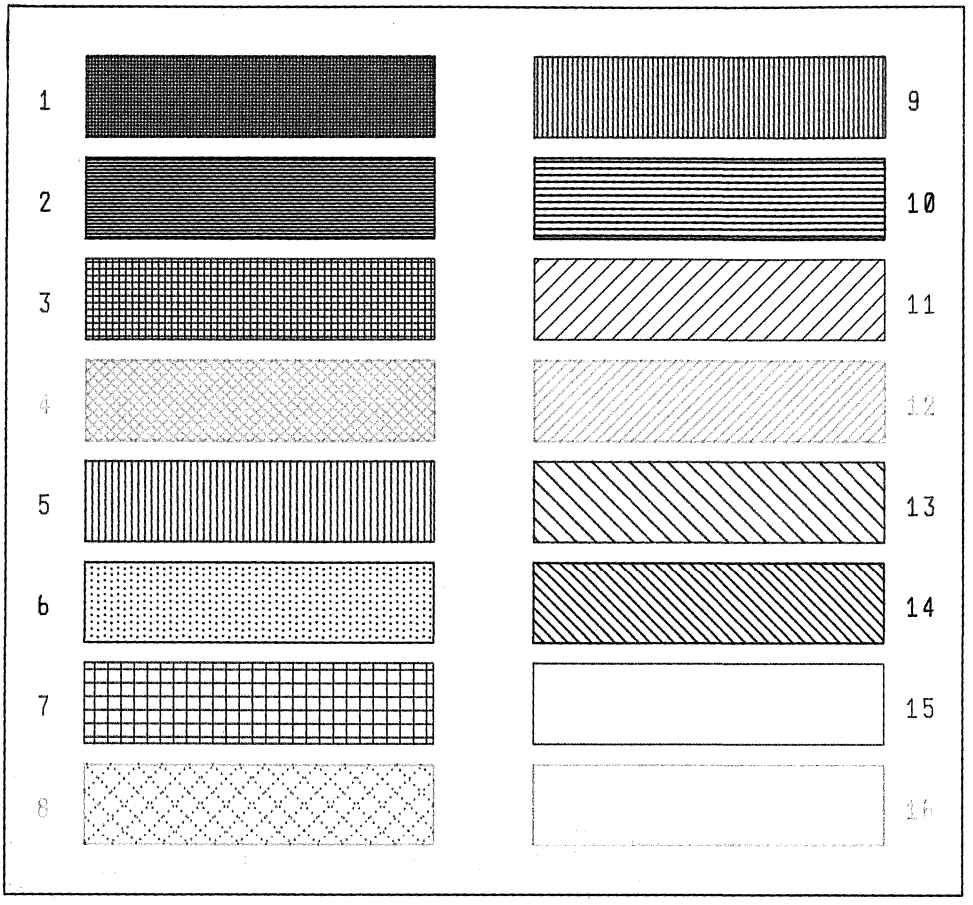
You can select one of 16 patterns for filled areas. All area-fills use the pattern you select until another pattern is selected. If you want a specific pattern for the area-fill (other than the default solid), you must assign a pattern before the GSAREA routine begins the area-fill. You can assign a different pattern after the GSENDA routine and before the next GSAREA routine.

The following routines are used for shading patterns:

GSPAT – Set the shading pattern. GSPAT specifies the pattern to be used when shading occurs. If a pattern is not specified, the shading is solid and of the current color.



Shading patterns. A code number is associated with each pattern. The code is used in the GSPAT routine to select the pattern.



Patterns as shown on a plotter. On the plotter, the lines used for the patterns are drawn closer together. Patterns on the printer are the same as those shown here for the plotter.

GSQPAT – Query the current shading pattern. GSQPAT returns the value of the current pattern being used.

Performance hint

Shading patterns are made up of many lines or dots. Each line or dot is processed separately and this takes some time. While you are debugging your program, you can comment out the GSAREA and GSENDA statements in your program to reduce the time it takes to process the picture.

Solid shading patterns on a plotter also take a long time to process. Use one of the other patterns to speed up the process.

How to Draw Graphics Symbols

Text shown by an OS/400 Graphics application program can be provided by GDDM or, when a program is written for use on a graphics work station, by the alphanumeric characters of a data description specification (DDS) display file. Plotters and printers can use only the characters provided in GDDM symbol sets.

The characters written by a DDS display file are the same characters you see written to the display screen in any other non-graphics application; they are always

of the same size and appearance. Here, these characters are called *hardware characters* because they are generated by the device.

Graphics text, on the other hand, is generated by GDDM. The characters in graphics text are called *graphics symbols*. With your program, you can control the placement, size, orientation, and type style (font) of the graphics text.

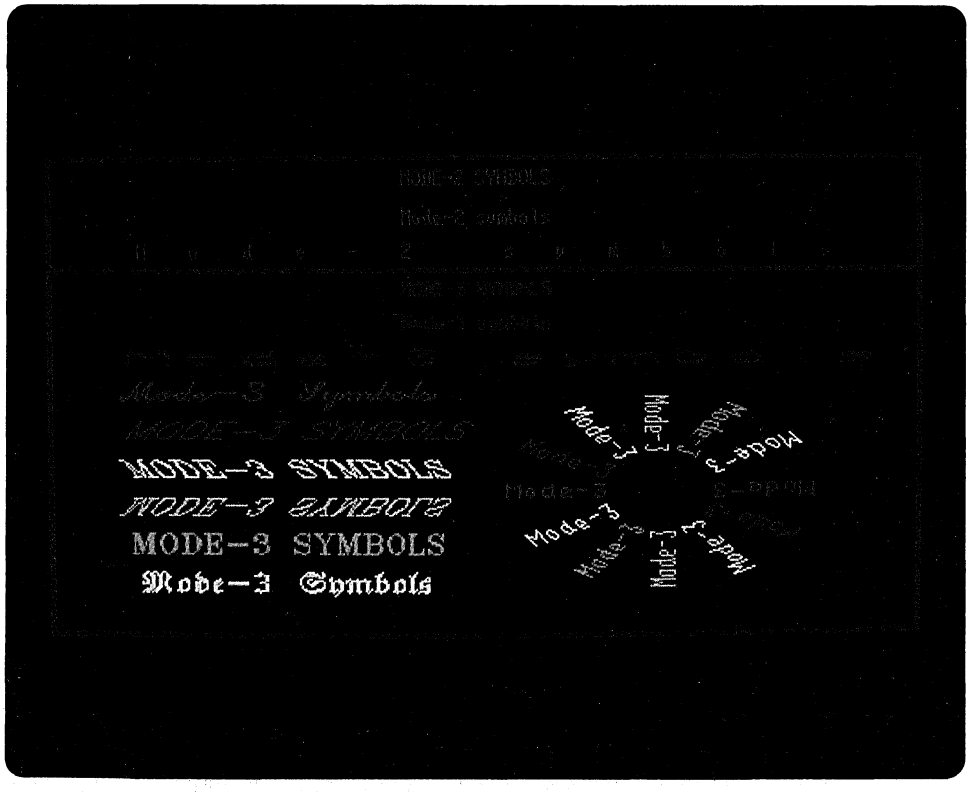
There are two types of graphics symbol you can use:

Mode-2, or *image symbols*
Mode-3, or *vector symbols*.

Image symbol characters are patterns of dots, while vector symbol characters are patterns of lines.

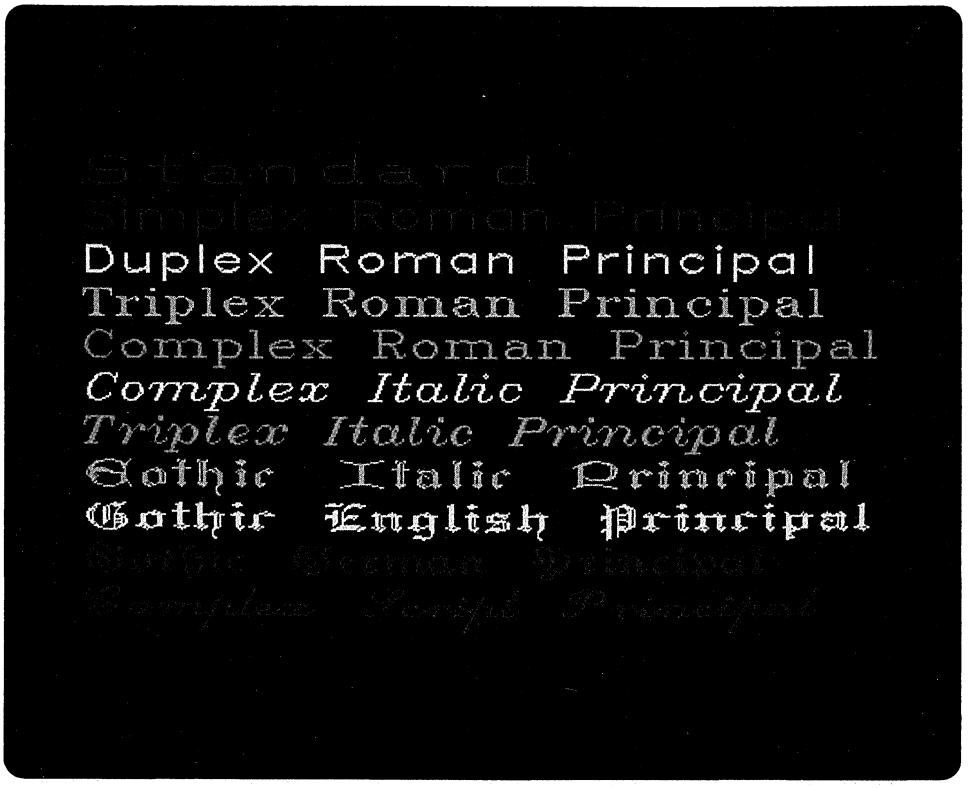
Image symbol characters are quite similar to *hardware* characters, except that some graphics characteristics apply. Image symbol characters respond to the current mode for color mixing, are drawn starting with the current position, and so forth.

Image symbol characters used on the printer respond to variations in printer file definition. For example, if a printer file is being used with a characters-per-inch (cpi) value of 15, rather than the default 10 cpi of printer file QSYS/QPGDDM, the characters are drawn closer together. See page 5-5 for details of this file.



Graphics symbols. Graphics symbols come in two modes: Mode-2 and Mode-3. Mode-2 characters are similar to *hardware* characters, while Mode-3 characters are *drawn* by GDDM.

Vector symbol characters are each small pictures drawn by GDDM. Vector symbols can be drawn with any size or orientation you choose. Text written with vector symbols can be shown in many different font styles.



Vector symbol font styles. Vector symbols can be drawn in many different styles.

All graphics symbol sets (both image and vector) are stored in OS/400 object type *GSS. For more information on object management of the *GSS object type, see "OS/400 Graphics Symbol Sets" on page 5-7.

A graphics symbol set must be loaded from the *GSS object, then selected for use in your program as the current symbol set (more than one can be loaded). The mode of the graphics symbol set can be specified also; for mode-3 symbols, attributes can be assigned that determine the way the symbols are drawn.

Controlling Symbol Sets

Graphics symbol sets that have been loaded apply only to the device your program is currently using; if the current device is closed by the DSCLS routine or is initialized again by the DSRNIT routine, the symbol sets must be loaded again for the new current device. Otherwise, the default symbol set is used. For information about device controls, see "Device Controls" on page 3-64.

The default symbol set depends on the type of application (GDDM or Presentation Graphics) and the device you are using. This table shows the symbol set used for each type of application program and each device type.

Current Device	GDDM	Presentation Graphics
Graphics work station	ADMMVSS (vector)	ADMMVSS (vector)
Plotters	ADMMVSS (vector)	ADMMVSS (vector)
3812 Printer (IPDS)	Uses printer hardware characters	ADMMVSS (vector)
3816 Printer (IPDS)	Uses printer hardware characters	ADMMVSS (vector)
4028 Printer (IPDS)	Uses printer hardware characters	ADMMVSS (vector)
4214 Printer (SCS)	ADMMISSG (image)	ADMMVSS (vector)
4224 Printer (IPDS)	Uses printer hardware characters	ADMMVSS (vector)
4234-2 Printer (SCS)	ADMMISSG(image)	ADMMVSS(vector)
5224 Printer (IPDS)	ADMMISSG(image)	ADMMVSS(vector)
5225 Printer (IPDS)	ADMMISSG(image)	ADMMVSS(vector)

All of the symbol sets shown here contain multinational characters and all have a similar appearance.

Notes:

1. Because image symbol set characters are constructed with a pattern of dots, mode-2 symbol sets explicitly loaded and selected for use on the plotter cause the plotter pens to strike the drawing surface repeatedly when these characters are drawn.
2. For IPDS Printers, the default symbol set under GDDM is the printer hardware character set selected by the CHRID parameter of the print file being used. For more information, see the CHRID parameter on the CRTPRTF command in the *CL Reference* manual.

Loading Symbol Sets

GSLSS – Load graphics symbol set. GSLSS loads a symbol set and makes it available for selection and use. Symbol sets are always loaded for the *current device* (the device being used). When a different device is selected for program output, the symbol sets you want must be reloaded.

Many symbol sets can be loaded for a particular device, but only one can be used at a time. Before you load a symbol set, library QGDDM must be in your library list.

If no symbol set has been loaded, the default symbol set (for the device) is used.

GSRSS – Release graphics symbol set. GSRSS releases a symbol set.

GSQNSS – Query the number of loaded graphics symbol sets. GSQNSS returns the number of symbol sets currently loaded for the device. This value can be used as the symbol-set-to-query parameter in the GSQSS routine.

GSQSS – Query loaded graphics symbol sets. GSQSS returns information about all symbol sets that have been loaded by GSLSS.

For a list of the names of the available symbol sets, see "OS/400 Graphics Symbol Sets" on page 5-7.

Selecting a Character Mode

GSCM – Set current character mode. GSCM sets the character mode for graphics symbols drawn by the program. If no symbol sets have been loaded and selected, the program uses the default symbol set for the mode specified. If no mode is specified, the default symbol set for the device is used. For the plotter, setting the character mode to 2 has no effect unless a mode-2 symbol set has been loaded and selected.

Selecting the Current Symbol Set

GSCS – Select a symbol set. GSCS selects a symbol set to be used as the current symbol set. The symbol-set identifier must name a symbol set that has been loaded from the *GSS object by the GSLSS routine.

If GSCM has not been used to specify a character mode, the default symbol set for the device is used.

If GSCM specified character-mode 3, and a symbol set is not specified or has not been loaded or if GSCS(0) is selected, the default ADMMVSS symbol set is used.

GSQCS – Query the current symbol set. GSQCS returns the value of the current symbol set.

Drawing Graphics Symbols

Graphics symbols can be drawn in the form of character strings.

Graphics symbols are drawn with their lower left corner over the current position. After a character string has been drawn, the current position is the position after the last character in the string.

The routines that draw graphics symbols of the current symbol set are:

GSCHAR – Draw a character string at a specified point. GSCHAR draws a character string at the position specified by x and y (instead of the current position).

GSCHAP – Draw a character string at the current position. GSCHAP draws a character string at the current position.

This routine is useful for continuing a character string at the point where a previous character string or primitive ended. GSCHAP is equivalent to a GSQCP followed by a GSCHAR that uses the values of the current position returned to the GSQCP routine.

Performance hint

Each mode-3 vector symbol is made up of many straight and curved lines. Each line is processed separately and the larger and more complex the character, the longer it takes to process. While you are debugging your program, you can comment out the GSCS statements in your program to reduce the time it takes to process the picture. The commented out statements prevent loaded symbol sets being used; the default ones are used instead.

Complex symbol sets on a plotter also take a long time to process. Use one of the simpler ones to speed up the process.

Attributes for Graphics Symbols

Graphics symbol attributes determine the symbol set, size, angle, direction, and shear of graphics symbol characters used in a picture. Because graphics symbols are drawn as small pictures, graphics symbol attributes have no effect on any alphanumeric characters (hardware characters) shown by a DDS display file in the same picture.

The graphics window (the coordinate system you specify) can have an effect on the attributes you specify for graphics symbols, especially vector symbols. If, for example, your coordinate system is set so that 2 units in the y-direction equal 1 unit in the x-direction, a *character box* (discussed next) set as x=1, y=2 will appear square, and the other attributes that determine the look of the symbols will also be affected.

Image symbols (mode-2) are similar in nature and appearance to hardware characters; graphics symbol attributes have little effect on them, except for spacing and placement. Graphics symbol attributes have the most effect on vector symbols (mode-3).

For mode-2 and mode-3 characters in graphics printer files, variations in the cpi (characters-per-inch) setting of the graphics printer file being used affect the horizontal spacing of characters in strings. For more information on graphics printer files, see "QPGDDM Printer File Considerations" on page 5-5.

Note: For both mode-2 and mode-3 characters, you must set the character mode with GSCM before any of the symbol-set attributes can have an effect on graphics symbols.

Setting the Graphics Symbol Size

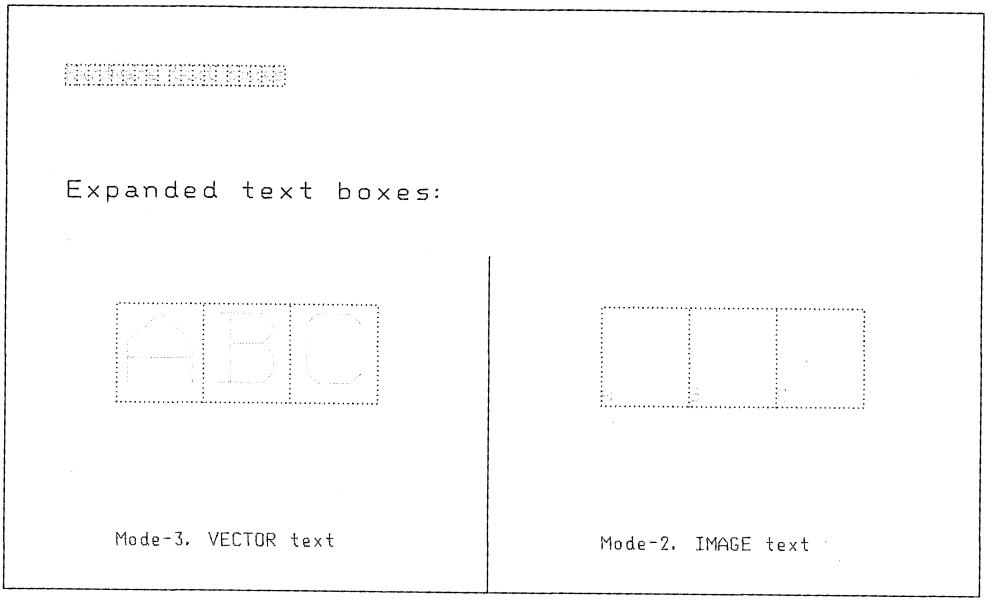
For vector symbols (mode-3), character size is determined by the attributes that specify the dimensions and coordinate system used by the program to position primitives on the screen (see "The Graphics Window" on page 3-53). Based on that system, you can specify the size of mode-3 characters in units of x- and y-coordinate values.

The *hardware cell size* is the size of each non-graphics, alphanumeric character (hardware character) used by the device, and of each image symbol (mode-2). The characters written by a DDS display file appear in hardware cell size. The hardware cell size for mode-2 graphics symbols on printers can vary with the definitions used for the page size or with the overrides that may affect the page size.

Each symbol is placed into an imaginary *character box*. For mode-3 vector symbol characters, the dimensions for each character are determined by the size of this character box.

If a graphics symbol size is not specified, the hardware cell size is used.

Character box. Hardware characters aren't affected by the character box size, while mode-3 symbols are enlarged to fill the box and mode-2 symbols are spaced according to the box width.




```
CALL GDDM ('GSCB',11.5,11.5)  ! Set character box
```

The routines for the graphic symbol size attribute are:

GSCB – Set character box size. GSCB sets the width and the height of the character box in x- and y-coordinate values.

GSQCB – Query current character box size. GSQCB returns the current values of the character box in x- and y-coordinate values.

GSQTB – Query text box. GSQTB returns the current values of the text box in x- and y-coordinate values. The *text box* is the long parallelogram formed by a string of character boxes. You set the text box when you use GSCHAR or GSCHAP to draw a string of graphics symbols.

GSQCEL – Query hardware cell size. GSQCEL returns the size dimension of the hardware cell in x- and y-coordinate values. These values vary with the coordinate system in use and the size of the graphics area in use; for example, creating a graphics page or field one half as wide as the default causes the value returned for the y dimension to double.

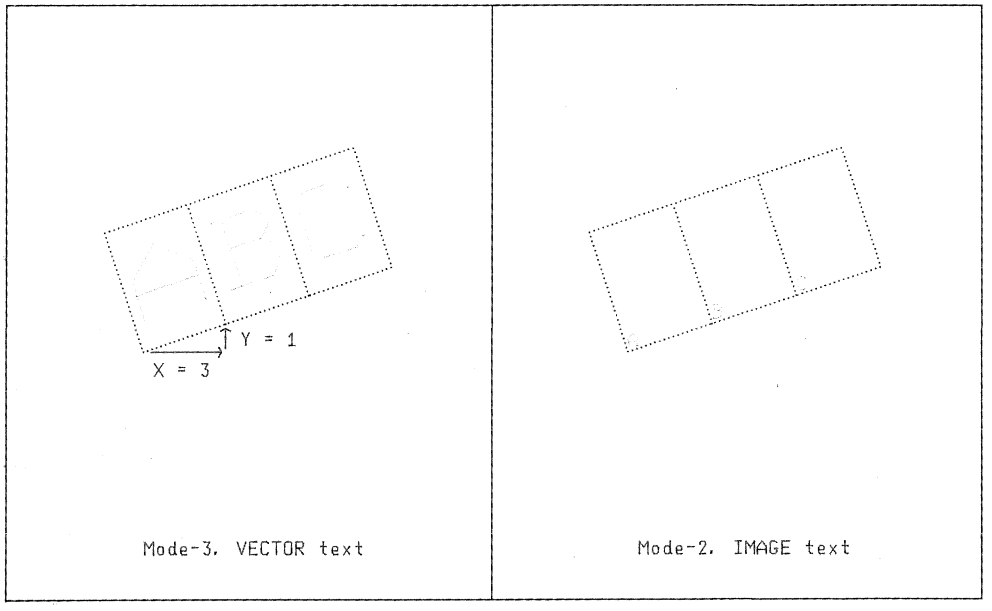
For printers, variations in the forms width and forms length values used in the printer file currently in use have an effect on the values returned in this routine.

Graphics Symbol Orientation: Character direction can be specified so that character strings are written on the screen (from the starting point) in any of four directions. Characters can also be rotated so that each character in a string is written along a baseline such that the string appears upside down, diagonally, or any orientation you choose. Characters can also be slanted (similar to a parallelogram). This attribute is called *character shear*. Like the size attribute, character shear depends on the coordinate system described for the entire picture; shear is specified in terms of x and y values.

Setting the Character Angle

The angle of the baseline can be set so that characters can be rotated in relation to the rest of the picture. Rotation values can be specified in terms of x- and y-coordinates, or by trigonometric operations.

Character angle. Here the string of adjacent character boxes (the text box) is rotated for mode-2 and mode-3 characters. Note that the baseline angle for the individual mode-2 characters remains horizontal.




```
CALL GDDM ('GSCA',3.0,1.0)  ! Set character angle
```

GSCA – Set character angle. GSCA sets the angle of the character baseline. If GSCA is not specified, a horizontal baseline is used.

If the coordinate system of the picture is such that x = y, then these three routines produce characters that are written on a 45-degree angle baseline:

```
CALL GDDM ('GSCA',1.0,1.0)
CALL GDDM ('GSCA',5.0,5.0)
CALL GDDM ('GSCA',12345.0,12345.0)
```

When x = y, a 45-degree baseline results. If you need a baseline of a specific angle that would be difficult to produce using the coordinate values of x and y, use the sine and cosine values of the angle radian.

To find the angle in radians to set a character angle use this algorithm:

```
DEGREE = 37                          !Character angle 37 degrees
RADIAN = DEGREE * (PI / 180)
DX = COS(RADIAN)                     !COS = cosine of angle
DY = SIN(RADIAN)                     !SIN = sine of angle
CALL GDDM ('GSCA',DX,DY)
```

Either of the values can be negative. The amount of rotation of the characters depends on the coordinate system established for the picture; for more information, see "The Picture Space" on page 3-48 and "The Graphics Window" on page 3-53.

The following routine is used to query the current character angle:

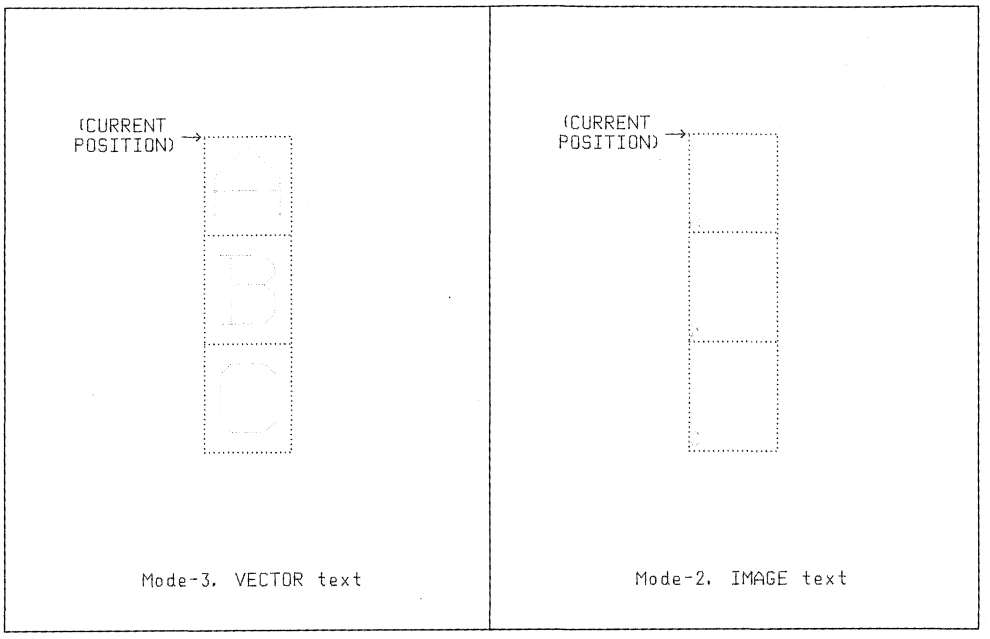
GSQCA – Query current character angle. GSQCA returns the x and y values used for the current character baseline angle. If sine and cosine were used for setting the angle, x and y are the radians.

Setting the Character Direction

Graphics symbol *direction* is the orientation of an entire character string. You can specify that the character strings are written from left to right (standard reading direction), top to bottom, right to left, or bottom to top. The character boxes can be tilted within the string by setting the character angle with the GSCA routine.

If a graphics symbol direction is not specified, character strings are drawn left to right.

Character direction. Here the character boxes are stacked for mode-2 and mode-3 characters. Character direction can also be set for bottom-up stacking and backwards-reading strings.




```
CALL GDDM ('GSCD',2)   ! Set downward-reading character direction
```

The following routines are used for setting and querying the character direction value:

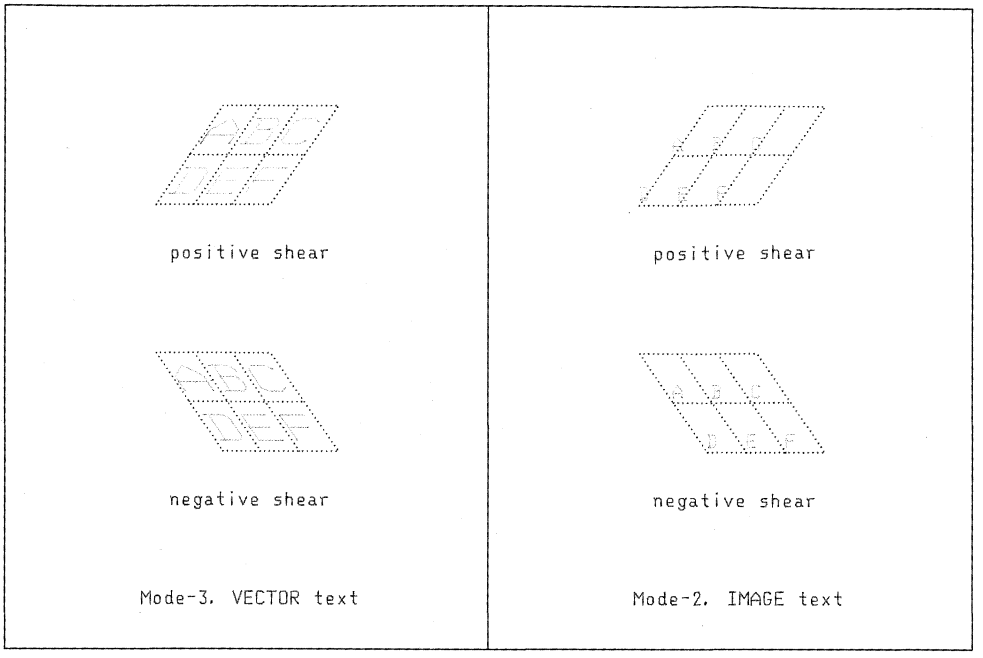
GSCD – Set character direction. GSCD specifies the direction of the character string.

GSQCD – Query current character direction. GSQCD returns the current direction value of the character string.

Setting the Character Shear

Graphics symbol *shear* gives each character a slanted, or italic, look. You specify shear with the same type of values used for the baseline angle. Shear values can be specified in terms of x and y coordinates, or by trigonometric operations.

The degree of shear of the characters depends on the coordinate system established for the picture; for more information, see "The Picture Space" on page 3-48 and "The Graphics Window" on page 3-53.



Character shear. Mode-3 characters are slanted by character shear, while mode-2 characters are only affected by the shift in position.

```
CALL GDDM ('GSCH',1.0,1.0)   ! Set positive shear

CALL GDDM ('GSCH',-1.0,1.0)  ! Set negative shear
```

These routines are used for setting and querying character shear values:

GSCH – Set character shear. GSCH sets the angle of character shear. The characters can slant to the left or the right, depending on the sign of one of the parameters used (right for positive or left for negative).

GSQCH – Query current character shear. GSQCH returns the values used to set the current angle of character shear.

Drawing Graphics Images

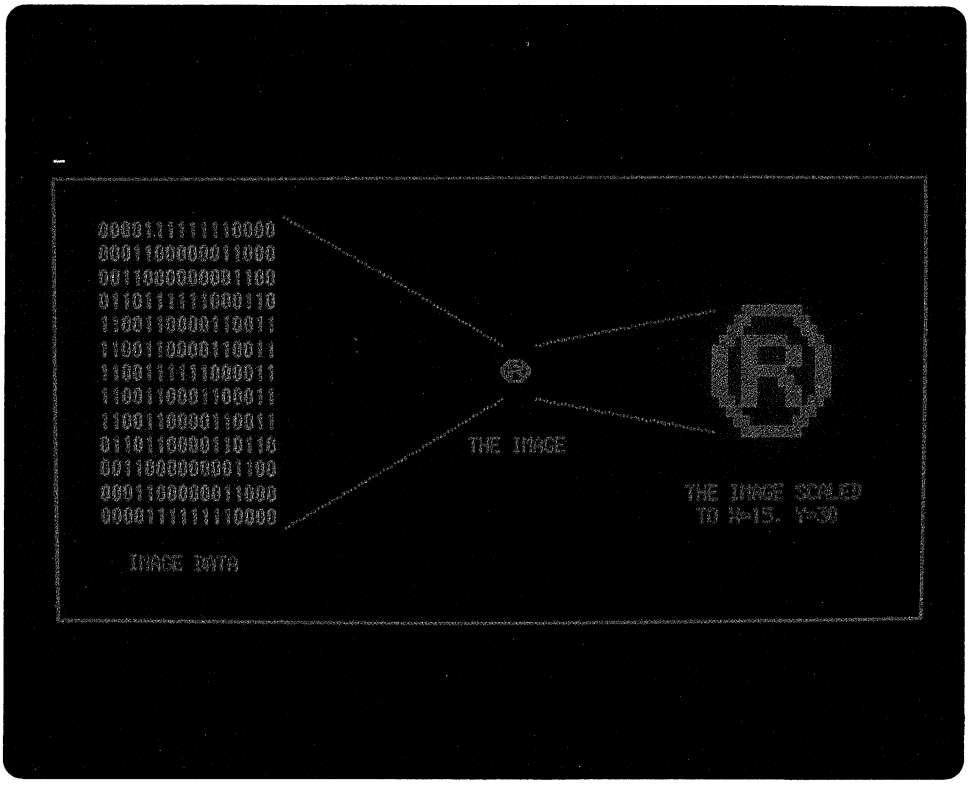
Another type of graphics primitive you can draw is the *graphics image*. A graphics image is a user-defined pattern of dots that can show a picture or symbol. You can define a graphics image in your program and then use it anywhere in your picture. The graphics image pattern is defined by a character variable string that represents a bit pattern of 0's and 1's; each 1 sets the associated pixel on, and each 0 leaves the associated pixel unchanged. The graphics image size is determined by the number of 0's and 1's (screen dots) that you include in the bit pattern.

Graphics images are drawn with the current color. You can draw multicolor graphics images by drawing one part of the graphics image in one color, and the rest of it in a different color (these separate parts have to be defined and called as separate graphics images and then overlaid).

Drawing a Graphics Image

GSIMG – Draw a graphics image. GSIMG draws a graphics image of a specified width and depth at the current position (the upper left corner of the graphics image is placed over the current position). The bit pattern defining the pixels of the graphics image is specified by a character string.

The character string that defines the graphics image can be formatted so that it contains rows and columns of 0's and 1's. The number of bits (0's or 1's) in the string must be a multiple of 8.



Graphics image and scaled graphics image. The graphics image bit pattern is converted to pixels. The graphics image can be scaled up, so that each display point is shown by a square.

The following program shows how a character variable can be used to define a graphics image in BASIC. (BASIC requires conversion from *binary* bit pattern data to hexadecimal notation.)

```
00010 REM ********************************************
00020 REM          INITIALIZE
00030 REM ********************************************
00040 CALL GDDM ('FSINIT')
00050 REM ********************************************
00060 REM          SPECIFY IMAGE DATA
00070 REM ********************************************
00080 REM 0000111111110000
00090 REM 0001100000011000
00100 REM 0011000000001100
00110 REM 0110111111000110            The bit pattern
00120 REM 1100110000110011            consists of 0's and
00130 REM 1100110000110011            1's.  Each 1 illuminates
00140 REM 1100111111000011            a PEL with the current
00150 REM 1100110001100011            color, while each
00160 REM 1100110000110011            0 leaves the display
00170 REM 0110110000110110            point as it was.
00180 REM 0011000000001100
00190 REM 0001100000011000
00200 REM 0000111111110000
00210 REM Convert the characters to hexadecimal as follows:
00220 REM 0000 1111 1111 0000 = HEX '0 F F 0'      where 0000 = HEX '0'
00230 REM 0001 1000 0001 1000 = HEX '1 8 1 8'            0001 = HEX '1'
00240 REM 0011 0000 0000 1100 = HEX '3 0 0 C'            0010 = HEX '2'
00250 REM 0110 1111 1100 0110 = HEX '6 F C 6'            0011 = HEX '3'
00260 REM 1100 1100 0011 0011 = HEX 'C C 3 3'            0100 = HEX '4'
00270 REM 1100 1100 0011 0011 = HEX 'C C 3 3'            0101 = HEX '5'
00280 REM 1100 1111 1100 0011 = HEX 'C F C 3'            0110 = HEX '6'
00290 REM 1100 1100 0110 0011 = HEX 'C C 6 3'            0111 = HEX '7'
00300 REM 1100 1100 0011 0011 = HEX 'C C 3 3'            1000 = HEX '8'
00310 REM 0110 1100 0011 0110 = HEX '6 C 3 6'            1001 = HEX '9'
00320 REM 0011 0000 0000 1100 = HEX '3 0 0 C'            1010 = HEX 'A'
00330 REM 0001 1000 0001 1000 = HEX '1 8 1 8'            1011 = HEX 'B'
00340 REM 0000 1111 1111 0000 = HEX '0 F F 0'            1100 = HEX 'C'
00350 REM ******************************************** 1101 = HEX 'D'
00360 REM          DRAW IMAGE                            1110 = HEX 'E
00370 REM ******************************************** 1111 = HEX 'F'
00380 DIM IMG$*26                     ! Image variable 1/2 hex string
00390 DIM X$*52                       ! Hex string variable
00400 X$='0FF01818300C6FC6CC33CC33CFC3CC63CC336C36300C18180FF0'
00410 IMG$=HEX$(X$)                   ! Convert back from hex to bits
00420 CALL GDDM ('GSMOVE',40.0,60.0)   ! Move current position
00430 CALL GDDM ('GSIMG',0,16,13,26,IMG$)
00440    ! Draw graphics image with 16 PELs across, 13 display
00450    ! points down, 26 bytes of PELs ((16*13)/8 = total)
00460 CALL GDDM ('GSMOVE',50.0,60.0)   ! Move current position
00470 CALL GDDM ('GSIMGS',0,16,13,26,IMG$,15.0,30.0)
00480    ! Draw scaled graphics image with 16 PELs across,
00490    ! 13 PELs down, 26 bytes of PELs
00500    ! ((16*13)/8 = total), scaled to 15 X-units for 30 Y-units.
00510 INTEGER ATTYPE,ATMOD,COUNT
00520 CALL GDDM ('ASREAD', ATTYPE,ATMOD,COUNT)
00530 CALL GDDM ('FSTERM')
00540 END
```

Drawing a Scaled Graphics Image

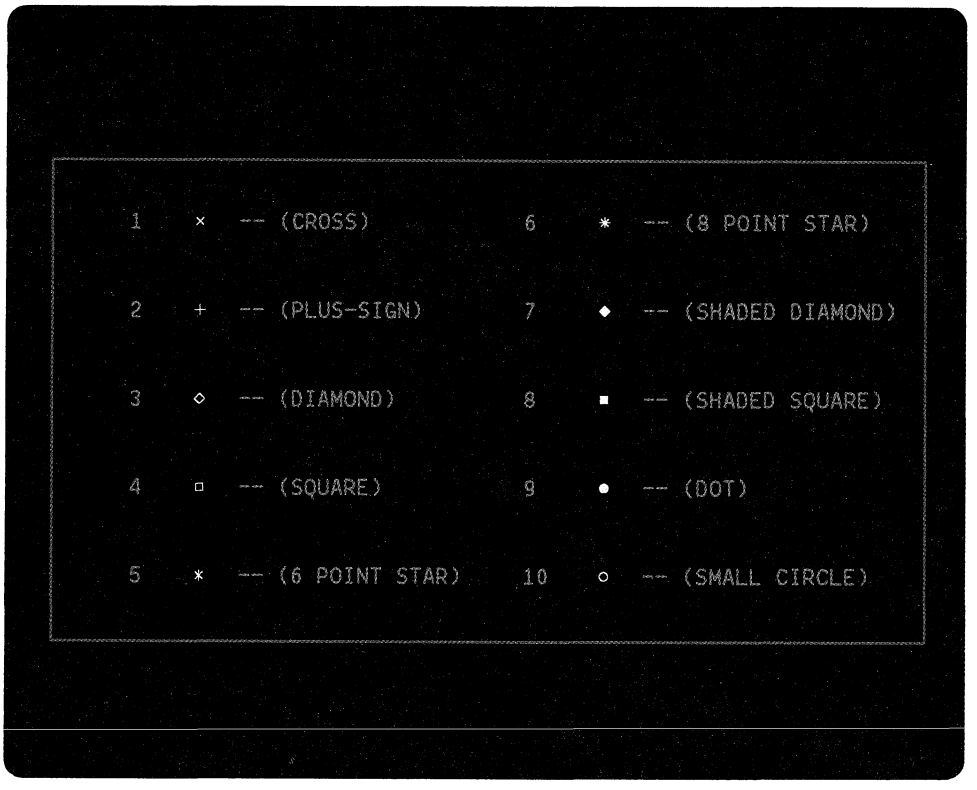
GSIMGS – Draw a scaled graphics image. GSIMGS performs the same function as GSIMG, except that you can specify an x and y range to enlarge the scale of the graphics image (in the current coordinate system); see "The Graphics Window" on page 3-53.

When the graphics image is scaled up, each PEL defined for the graphics image is represented on the picture by a square. Graphics images cannot be scaled to a size smaller than the defined GSIMG graphics image.

For examples of programs that draw graphics images in the other high-level languages, see Chapter 6, "Graphics Application Program Examples."

How to Draw Markers

You can use the graphics primitive marker to highlight points on charts or pictures.



Markers. Ten marker types are available.

Loading Marker Symbol Sets

GSLSS – Load graphics symbol set. GSLSS loads a marker symbol set and makes it available for use.

Only one marker set may be loaded at one time for a particular device, and once loaded, the marker set is automatically selected for use as the current marker set.

If no marker set has been loaded, the default marker set is used.

GSRSS – Release graphics symbol set. GSRSS releases a marker symbol set.

Selecting a Marker

GSMS – Select the marker symbol. GSMS specifies the marker symbol to be used. If a marker symbol is not specified, an x (cross) is used for the current marker symbol type.

GSQMS – Query the current marker symbol. GSQMS returns the value of the current marker symbol.

Drawing Markers

GSMARK – Draw the marker symbol. GSMARK draws the current marker symbol, placing the center of the marker symbol at the coordinate specified by x and y.

GSMRKS – Draw a series of marker symbols. GSMRKS draws a series of current marker symbols at points specified by arrays of x- and y-coordinate values.

Drawing Scaled Markers

GSMSC – Set the marker symbol scale. GSMSC specifies the size of the marker symbol to be used. The marker symbol size is specified in units of the size used for the coordinate system in use for the picture. If a marker symbol scale is not specified, the default marker size is used. Markers cannot be scaled to a size less than the default marker size.

GSQMSC – Query the current marker symbol scale. GSQMSC returns the value of the current marker symbol size.

Drawing Pictures: Summary

The part of this chapter you have just finished reading (the first part of Chapter 3, "Using GDDM") showed you the GDDM primitives and their attributes used for all pictures (and charts, for Presentation Graphics) created by OS/400 Graphics.

You learned about the GDDM primitives, which are:

Lines
Area-fills
Graphics symbols
Graphics images
Markers

Lines define forms and shapes that, when enclosed, can be filled with patterns of color by the area-fill. Graphics symbols come in two varieties, mode-2 image symbols and mode-3 vector symbols. Graphics symbols provide text for pictures. Graphics images allow you to define your own graphics image symbols, except that the graphics image you define cannot be used as text in a graphics symbol routine, and you must process the image each time you use it. Markers can be used to highlight and pinpoint areas in a picture.

All of these primitives can have attributes assigned to them. All primitives have one common attribute, color, but most attributes are unique to a primitive. For example, a wide line-width attribute for line drawing routines has no effect on graphic symbols. All attributes do, however, have a common characteristic: once an attribute is specified, it stays current and in use for all applicable routines until it is changed. For example, when you set the color attribute to red, *everything* will show up in red until the color attribute is changed. When you set the line type to dotted line, every line will be drawn dotted until you change the attribute. This characteristic of attributes is called *the current mode*.

The current mode characteristic also applies to the GDDM control routines that are discussed in the next section, Controlling Graphics, where you will read about the GDDM routines you can use to control such things as:

Whether the graphics environment is initialized (whether the AS/400 System is expecting calls to other GDDM or Presentation Graphics routines).

What actions are taken when an error occurs in the graphics program.

Which display device is the current one and when does the picture get sent there.

Which of the pictures drawn by the program is shown.

How much of the screen is used to show the picture.

How much of the picture is shown.

The controls described next can be used for more versatile, more powerful OS/400 Graphics programs.

Controlling Graphics

In the first part of this chapter, you learned that some GDDM routines specify what is drawn on the picture and where it is drawn (primitives), and that others specify how those primitives should look (attributes).

This part of the chapter describes the GDDM routines that specify control functions for the graphics program and the control of devices. The program and the devices and their characteristics are all part of the *graphics environment*.

Program Controls

Program controls initialize and terminate the graphics environment and determine the action taken when errors are encountered.

Graphics Environment Controls

The graphics environment controls signal to the system whether it can expect calls to graphics routines.

Initializing and Terminating the Graphics Environment

FSINIT – Initialize the graphics environment. FSINIT initializes the graphics environment. FSINIT must be the first GDDM routine called in the program.

FSTERM – Terminate the graphics environment. FSTERM terminates the graphics environment. All storage used by the graphics environment is freed. FSTERM must be the last GDDM routine called in the program; if any other GDDM routines follow the FSTERM routine, they are ignored unless they are preceded by an FSINIT routine.

FSRNIT – Reinitialize the graphics environment. FSRNIT reinitializes the graphics environment. FSRNIT is equivalent to using the FSTERM routine followed by the FSINIT routine, except that the graphics environment retains the information that is available about the current device. For more information about device control routines, see "Device Controls" on page 3-64.

Error Handling Controls

Routines are available in your graphics program for handling and querying errors. You can specify the error-handling program you want executed when an error occurs.

Specifying an Error Handling Program

FSEXIT – Specify an error exit program and threshold. You can use FSEXIT to specify a program that performs error handling when an error is encountered with a routine. The severity of the error must equal or exceed the severity threshold (also specified by FSEXIT). The error-handling program can be written to terminate graphics, or it can return control to the original program. Control is returned to the routine that follows the routine that caused the error.

For example, a program that is called after a user enters an invalid value (for a program that uses subfiles to prompt the user for values that are passed to GDDM routines) is an error-handling program. The program processes the error by prompting the user for the correct value, then passes the corrected routine back to the program, which resumes execution. You do not need to include error-checking statements in your program, use the error-checking capability of GDDM.

When a severity threshold is specified for FSEXIT but no error recovery program is specified, an error that equals or exceeds the threshold causes message CPF8619 to be issued. In that situation the MONMSG (Monitor Message) CL command can be used to control error handling.

Querying the Last Error

FSQERR – Query last error. FSQERR returns information about the last GDDM error. The information is placed into the named error record. The error record can be up to 569 bytes long. If no error has occurred in the program, the default information is returned.

For more information on error handling, see "Error Recovery" on page 5-13.

Display Controls

Display controls send the picture or updates of the picture to the display station or the plotter. (The destination depends upon the device control routine specified; for more information, see "Device Controls" on page 3-64 and Appendix A, "Devices Compatible with the AS/400 System.")

Sending the Picture to a Device

FSFRCE – Update the picture. FSFRCE sends to the device all changes that the program has made to the picture since the last ASREAD or FSFRCE routine. FSFRCE does not require a response from the device (unlike ASREAD).

FSFRCE updates the display and immediately returns to the program. Because FSFRCE does not require a response, program actions can change the picture before the operator of the program sees the updated display. FSFRCE is useful in a program just *before* an alphanumeric write/read operation where the graphics picture should be overlaid by alphanumerics.

If the program erases parts of a picture that is being displayed, FSFRCE redraws the entire picture. If parts are being added, only the new parts are drawn. On plotters, an FSFRCE or ASREAD causes everything within a graphics segment to be drawn again. On printers, a page eject occurs after every FSFRCE or ASREAD and everything within a graphics segment is drawn again).

ASREAD – Device input/output. ASREAD performs an FSFRCE update to the display, then unlocks the work station keyboard and waits for operator acknowledgement. The type of acknowledgement to the display is returned to the program (Enter key, F keys, or Clear key).

You can use this returned value to alter the flow of control in your program. For the plotter and the printer, no user response is returned.

FSREST – Retransmit the picture. FSREST causes the picture to be deleted and completely redrawn the next time the ASREAD or FSFRCE routine is used.

For a program that sends the picture to the plotter or printer, ASREAD and FSFRCE provide identical function, but a low-severity error is issued for the ASREAD routine.

Picture Controls

Every program that uses GDDM primitive and attribute routines to draw a picture also uses GDDM control routines to select the device used to display the picture, define the size of the picture, and the coordinate system in use. If the program contains routines to construct several different pictures, GDDM picture control routines select which picture to show. If several pictures are shown at the same time, GDDM picture control routines position them on the screen.

The simple programs in Chapter 2, "The Application Program Interface to Graphics" that drew the picture of the envelope and the line chart used all of these controls, but you did not see them in the program; the control routines assumed default values. Simple graphics programs have the following characteristics, which result from default control values:

One picture is defined and shown per program.

A coordinate system of 0 through 100 is used for both the x and y ranges.

Both the x and the y ranges span the entire width and depth of the screen.

The entire screen is used to show the picture.

The picture is shown on the device that called the program.

If any of your programs will specify further controls for the picture, then you should understand the *graphics hierarchy*.

The graphics hierarchy comprises:

1. The device
2. The page
3. The graphics field
4. The picture space
5. The viewport
6. The graphics window
7. The graphics segment

The Device

The device is at the top of the hierarchy; it is the destination to which the output of the graphics program is sent. Valid destinations for pictures are:

- IBM personal computer with work station function (WSF)
- IBM personal computer with work station emulation (WSE)
- 5292 Model 2
- IBM personal computer with 5250 emulation
- IBM 6180 Plotter
- IBM 6182 Plotter
- IBM 6184 Plotter
- IBM 6185 Plotter
- IBM 6186-1 Plotter
- IBM 6186-2 Plotter

- IBM 7371 Plotter
- IBM 7372 Plotter
- IBM 4214 Printer
- IBM 4234-2 Printer
- IBM 5224 Printer
- IBM 5225 Printer
- IBM 3812 Printer
- IBM 3816 Printer
- IBM 4028 Printer
- IBM 4224 Printer

Graphics output for a printer can be sent directly from the application program to the printer or it can be placed onto an output queue as a spooled file. However, a spooled file created for one type of printer will generally be incompatible with another type of printer. A program can send its output to any identified device.

Device control routines can be used to select an output device and to identify characteristics of the device to the program. For a description of device control routines, see "Device Controls" on page 3-64.

The Page

Some graphics programs are written such that they use GDDM (or Presentation Graphics) routines to construct a picture, send the picture to a display station or a plotter, and then end. Other more complex programs construct several different pictures; each of these pictures can be sent to a device one at a time, or they can be sent to different devices, one at a time. The method GDDM uses to keep track of these individual pictures is called the *page*.

Each picture created by a program is considered to be one page of information. Each page has a number assigned to it. A program can construct a picture for a first page and then continue on to construct a completely different picture (a second page) without having sent the first page to a device. Even though the first picture has not been displayed, its segments have been processed (constructed) and the picture is stored in the system, waiting to be selected and sent to a device. (Segments for a page are discussed later.) Alternatively, a program can send each page to a device as soon as the picture has been processed.

The number assigned to a page when it is created identifies it in GDDM routines where a page is selected, cleared, deleted, and queried.

The program sends a page to a device when an ASREAD or FSFRCE routine is encountered:

If several different pictures (pages) have been constructed by the program, the ASREAD or FSFRCE sends to the device the *current* page. The current page is one that has been *selected*. To send one of the other pages to the device, the program selects that page. The newly-selected page is now the current page, and it is sent to the device as soon as an ASREAD or FSFRCE routine is encountered.

A page can be selected again and sent to the device, or the program can add more graphics to it and then send it to the device. When a page is *cleared*, it still exists in the system, but all the attributes and picture characteristics defined for it are now the default values.

A page that has been *deleted* ceases to exist. If the program deletes a page and then later selects the number of the deleted page, the system creates a new

current page with that identifier (with default attributes and picture characteristics).

An ASREAD or FSFRCE routine in a page sent to a printer causes a *page eject*; that is, after the current page has printed, the printer positions a clean form for the next print operation.

An ASREAD or FSFRCE routine in a page sent to a plotter with form feed enabled causes a *page eject*; after the current page has plotted, the plotter loads a new sheet of paper for the next plot operation.

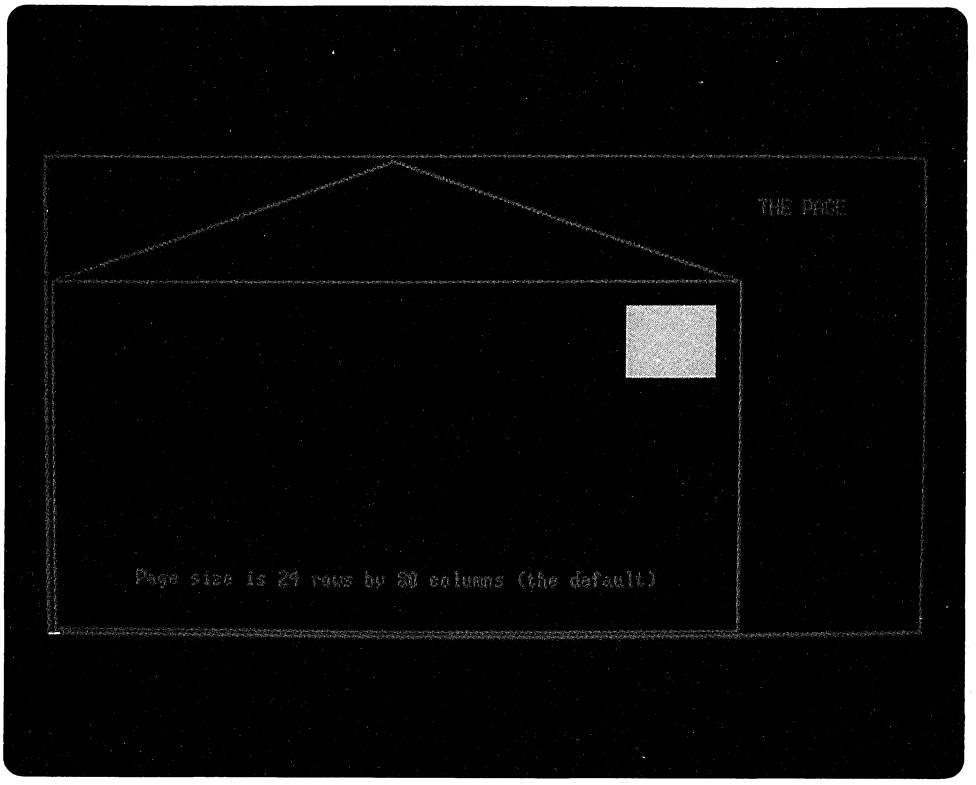
If a program that defines several pages is to send some of the pages to a different device, the program must create pages specifically for that device. For example, a program that sends PICTURE1 to both a work station *and* a plotter, must create two pages. Each page contains all the same drawing routines, but each must be associated with a particular device.

In the program, creating the page and the routines that draw the picture for the page must follow the device control routines (described later). An ASREAD or FSFRCE sends the page to the current device. Then, to send the picture to a different device, the program must select that device as the current one and create a new page with the same drawing routines, followed by the ASREAD or FSFRCE to send that page to the device.

Creating a Page

FSPCRT – Create a page. FSPCRT assigns a number to the page. You can use the number in other routines to select, query, or delete the page later in the program. The default page number is zero, which is assigned when the program is initialized by FSINIT or reinitialized by FSRNIT.

A page is specified in terms of the physical rows and columns it uses on the device (the rows and columns are ignored for the plotter).



Page. The default page uses the entire graphics portion of the screen.

```
CALL GDDM ('FSPCRT',1,24,80,0)
           ! Create page 1, 24 rows, 80 columns, type 0
```

If a page is not explicitly created for the graphics work station by your program, a default page is created with the dimensions of the physical boundaries of the screen, 24 rows by 80 columns. You can create a page with less than 24 rows and 80 columns, but no graphics can be shown outside the physical row and column limits of the page on the screen.

For a program that uses a printer, the default page size is that specified by the graphics printer file currently in use. For the default printer file, the default page is 90 rows by 132 columns (90 lines by 132 characters wide on the printer page), at 10 characters-per-inch (cpi). If you change the printer file forms width or overflow value, or define your own printer file, the default page assumes the dimensions of that printer file. A page cannot be created with row and column values that exceed the dimensions of the current device or current printer file.

Selecting a Page

FSPSEL – Select a page. FSPSEL selects a page to be the current one. This page remains current until it is deleted or another page is selected or created. An ASREAD or FSFRCE sends the contents of the current page to the current device.

Clearing a Page

FSPCLR – Clear the current page. FSPCLR clears all graphics fields from the current page. The page exists as if it were being newly created; no graphics field, picture space, viewports, or graphics windows exist. (All of these other elements in the graphics hierarchy are explained later.)

Deleting a Page

FSPDEL – Delete a page. When FSPDEL deletes a page, any graphics segments associated with the page are also deleted. (Graphics segments are explained later.)

Querying Page Information

FSPQRY – Query the page. FSPQRY returns information about the specified page, including default values if the page was created by default.

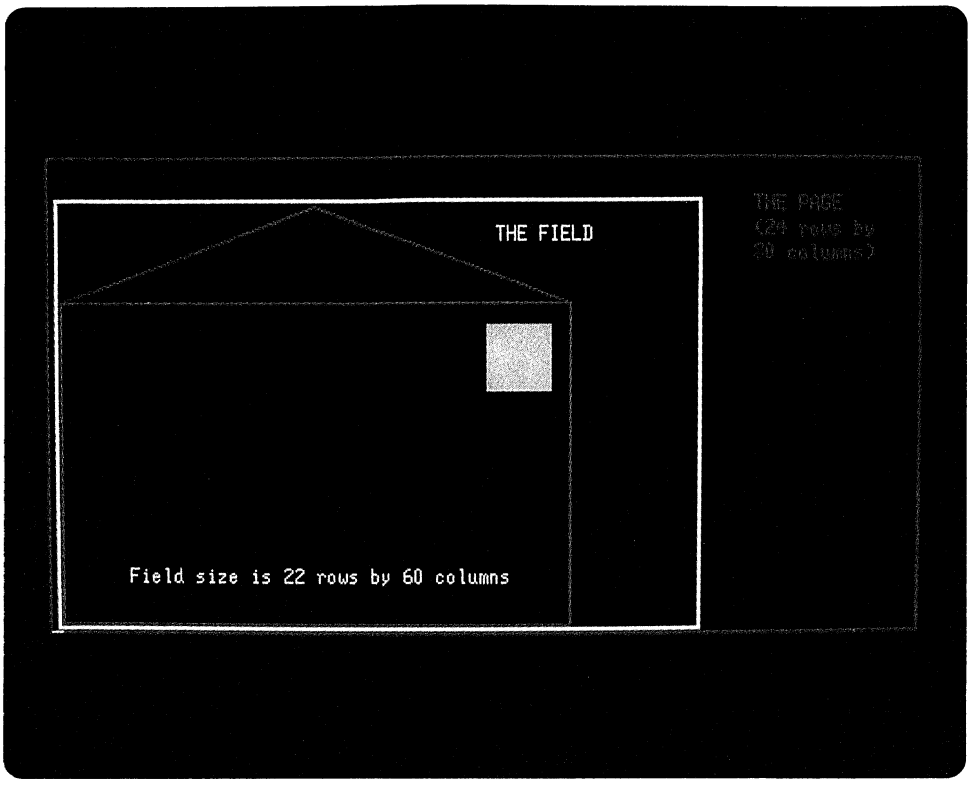
For the printer, the size of the page is returned. If FSPCRT has not been specified in the program, the default printer page is defined by the graphics printer file in use; however, the values returned to the FSPQRY routine may not match those of the current graphics printer file if a lines-per-inch (lpi) value other than 9 is specified. For more information on how the default page dimensions are calculated see "QPGDDM Printer File Considerations" on page 5-5.

FSQCPG – Query the current page. FSQCPG returns the number of the current page.

FSQUPG – Query a unique page number. FSQUPG returns the unique number of an unused page. You can use this routine to avoid using a number that is being used for an existing page; if you create a page with the number of another page, an error occurs.

The Graphics Field

You can use the graphics field to define an area of the page where the program's picture appears. The graphics field is specified by the position of its top left corner and its depth and width, using row and column numbers.



Graphics field. The graphics field further defines the area of the screen for display of the picture.

```
CALL GDDM ('GSFLD',2,1,22,60)
          ! Define field starting with row 2, column 1,
          ! 22 rows deep, 60 columns wide
```

Graphics fields can be defined and cleared. If a graphics field is not specified, it defaults to the size of the page. The graphics field is defined for, and applicable to, the current page only. If an existing graphics field is redefined, the existing contents of the page are lost, and all graphics segments in the original field are deleted. (Graphics segments are explained on page 3-58.)

Defining a Field

GSFLD – Define the graphics field. GSFLD defines the graphics field for the current page. A graphics field cannot be defined to exceed the dimensions of the current page.

Clearing a Field

GSCLR – Clear the graphics field. GSCLR clears the graphics field and deletes all segments.

The Picture Space

The picture space defines the ratio of the width of the picture to its depth, within the graphics field. The picture space can be used to ensure the ratio of one side of your picture to the other. For example, because the ratio of the default graphics field in the default page for the 5292 Model 2 is 1 : 0.529663 (one half as deep as it is wide), you can set the picture space to ensure the outside dimensions of the picture are an exact ratio.

Defining a Picture Space

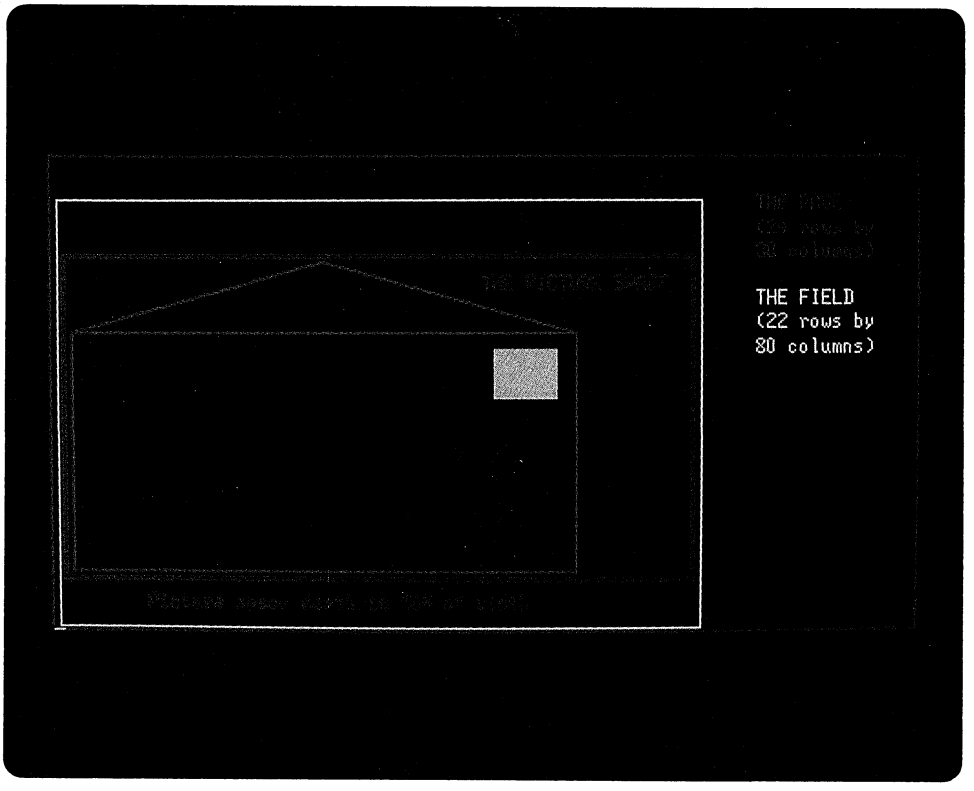
GSPS – Define the picture space. GSPS specifies the picture space to be used in the current graphics field, in terms of the ratio of the dimension of one side to the other. If you do not specify a picture space, the dimensions of the graphics field are used. Use GSQPS to see what the dimensions are.

The picture space defines the ratio of the width of the picture to the height. For example, if your program will be drawing a floor plan of a building that measures 25 meters by 50 meters, the ratio of the picture could be specified as:

```
CALL GDDM ('GSPS',1.0,0.5)
```

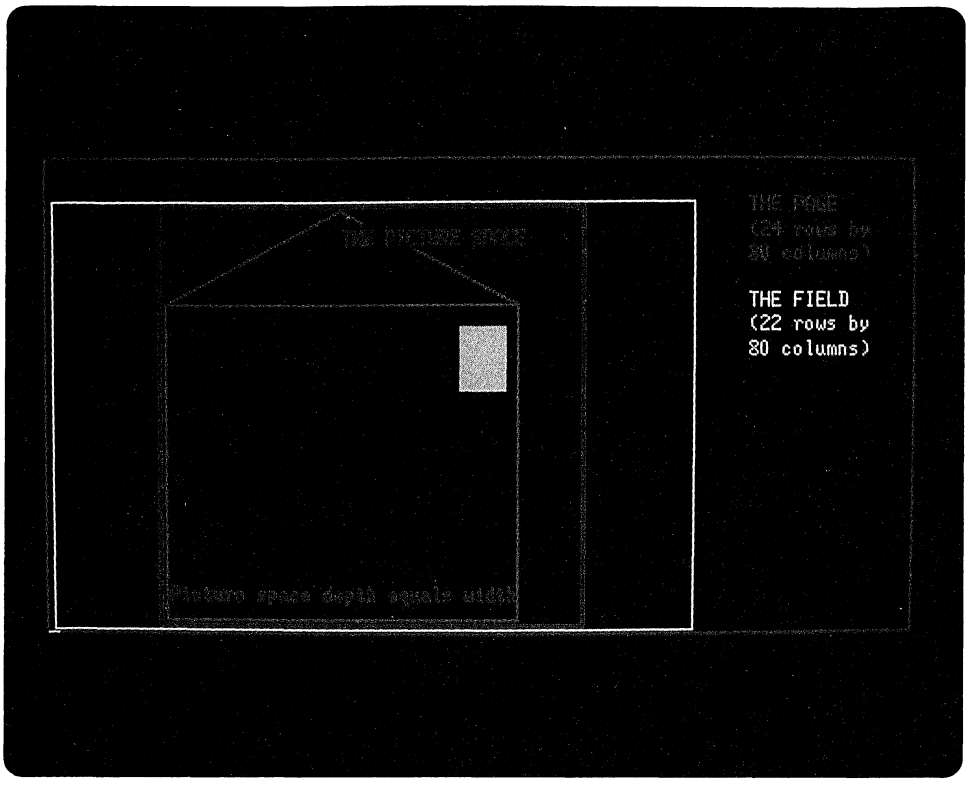
In this case, the width (50 meters) of the picture is twice that of the depth (25 meters); width = 1.0, depth = 0.5. One of the values used in GSPS must be 1, and the other less than or equal to 1.

The picture space is placed into the graphics field so that its largest dimension is equal to one of the dimensions of the graphics field. The picture space is centered in the graphics field.



Wide picture space. The depth of this picture space is one half the measure of the width.

```
CALL GDDM ('GSPS',1.0,0.5)  ! Set the picture space
```



Square picture space. The depth of this picture space equals the width.

```
CALL GDDM ('GSPS',1.0,1.0)  ! Set the picture space
```

GSQPS – Query the picture space. GSQPS returns the value of the picture space definition, in terms of the ratio of one side to the other.

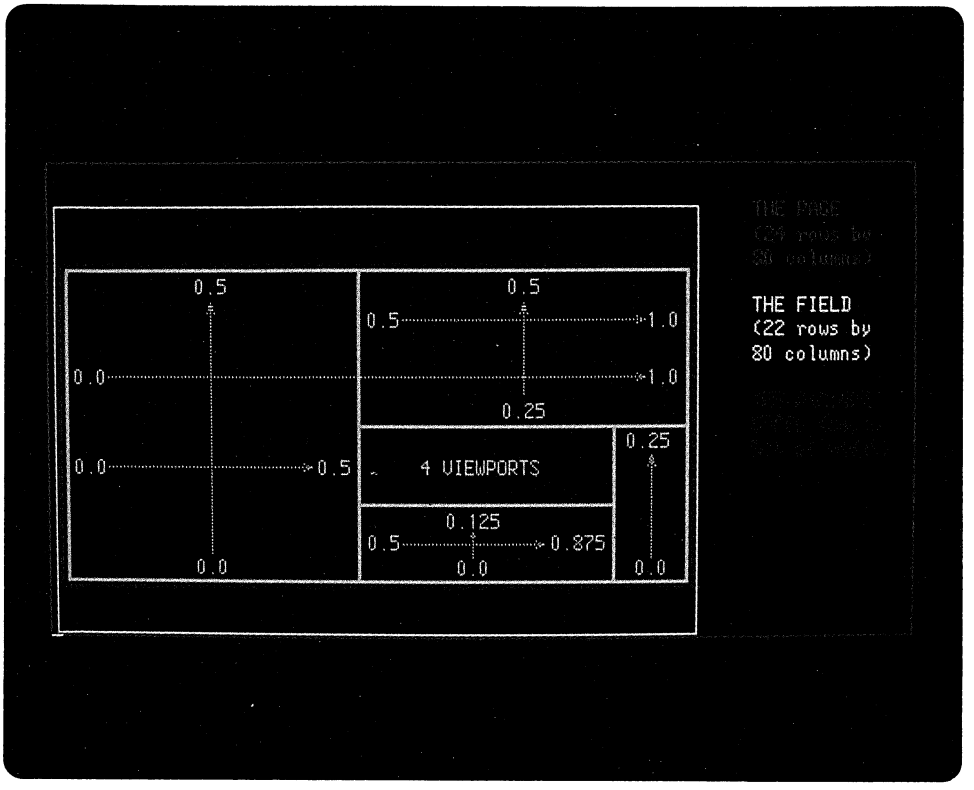
The Viewport

You can use the viewport to show individual pictures in different parts of a single picture area. The graphics field and the picture space are elements in the hierarchy that define the location and dimensions on the page of the picture when it is displayed. Only one graphics field and picture space can be specified per page, but more than one viewport can be specified.

The *picture space* defines *where* on the graphics field the picture originates, as well as the *ratio* of the picture's dimensions. The *viewport* specifies *where* on the picture space the picture is shown. If you do not specify a viewport, the entire picture space is used.

A viewport does not have to be specified in your program unless you want to show two pictures at the same time. To do that, your program must:

1. Create a page for the current device.
2. Specify a viewport for the graphics field and picture space (the graphics field and the picture space can be defined or allowed to default), and then construct the first picture.
3. Specify a new viewport, and construct the second picture.
4. Send the page to the current device with ASREAD or FSFRCE.

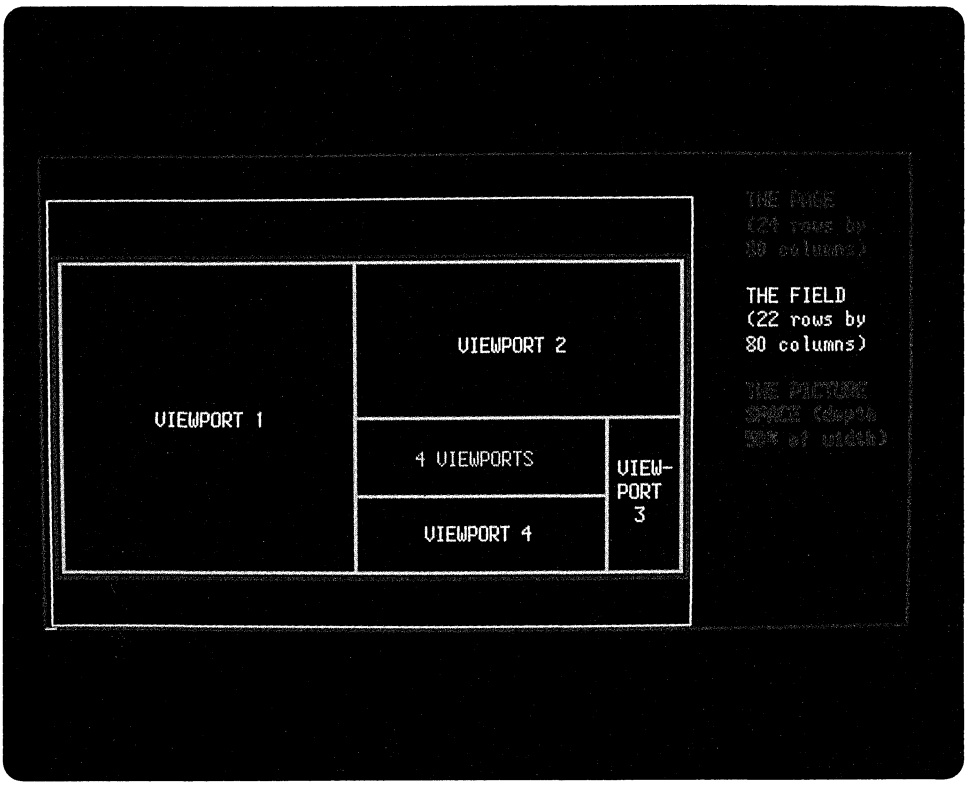


Values used to set the viewport. The viewport is defined with floating-point values that correspond to the values used to set the picture space.

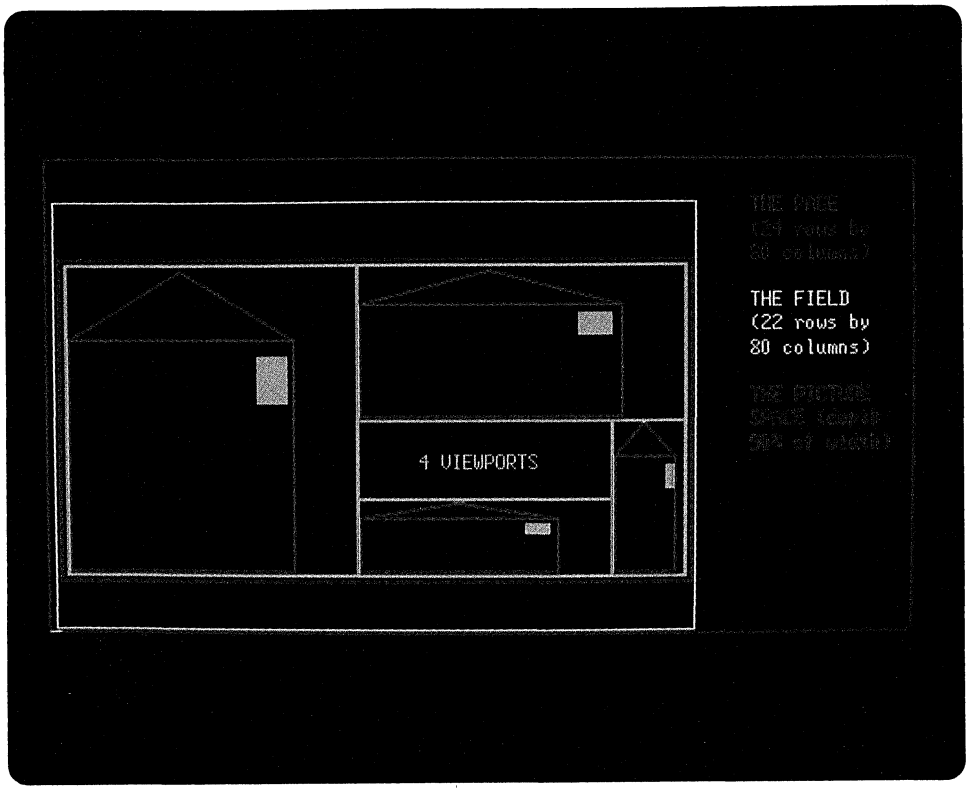
Defining a Viewport

GSVIEW – Define a viewport. GSVIEW specifies the location of viewport boundaries in picture-space units. If you do not specify a viewport, the dimensions of the picture space are used.

Viewport. Each viewport is defined with values that correspond to the values of the picture space.




```
CALL GDDM ('GSPS',1.0,0.5)
! Picture space right border 1.0, top border 0.5
   .
CALL GDDM ('GSVIEW',0.0,  0.5,   0.0,    0.5)
! Set viewport 1  = left, right, bottom, top
! in the same units used for the picture space
   .
CALL GDDM ('GSVIEW',0.5,  1.0,   0.25,   0.5)
! Set viewport 2  = left, right, bottom, top
   .
CALL GDDM ('GSVIEW',0.875,  1.0,    0.0, 0.25)
! Set viewport 3  = left, right, bottom, top
   .
CALL GDDM ('GSVIEW',0.5,  0.875,    0.0, 0.125)
! Set viewport 4  = left, right, bottom, top
```



Viewports with pictures. Each viewport defined can show an individual picture.

Note: You cannot use a viewport to position a Presentation Graphics chart; in Presentation Graphics programs, the counterpart to the GSVIEW routine of GDDM is CHAREA.

A viewport can be redefined at any point (except within a segment or area-fill) in the program. If viewports are not specified, a single viewport is used that has the boundaries of the picture space.

This routine can be used to query the current viewport:

GSQVIE – Query the viewport. GSQVIE returns the values of the current viewport boundaries in picture-space units.

The Graphics Window

The graphics window specifies the extent and range of the coordinates used by graphics primitive routines to draw a picture on the viewport. Because the coordinate system used by graphics primitives can be much smaller than the range of coordinates defined by the window, this coordinate system is sometimes referred to as the *world coordinate* system.

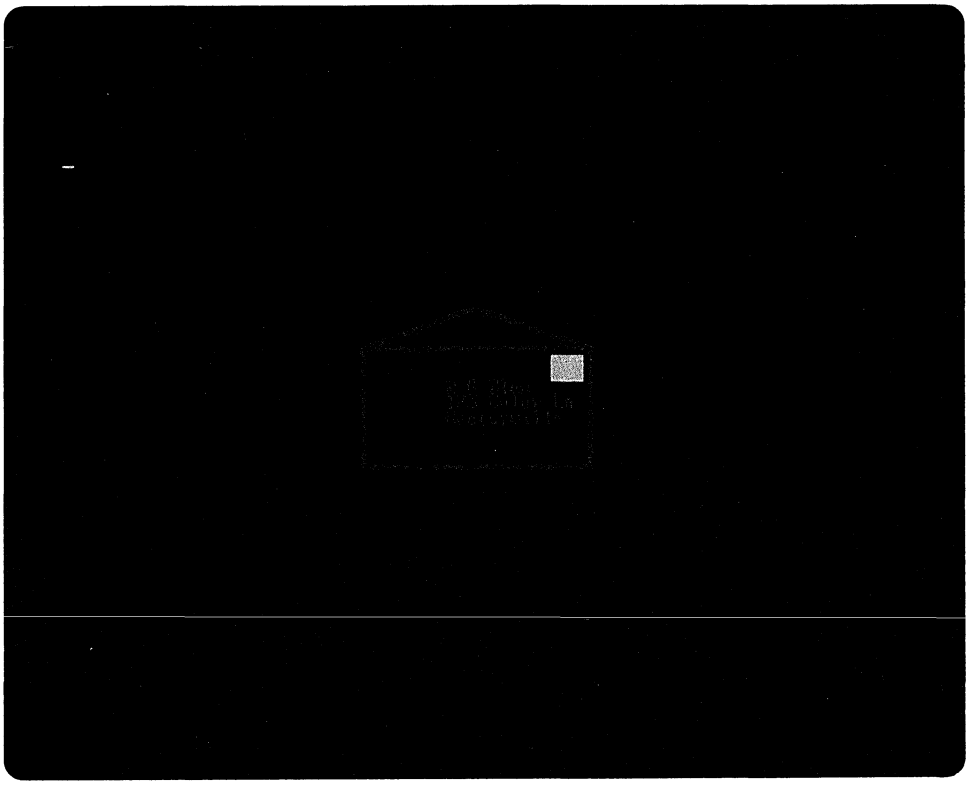
The items in the graphics hierarchy described to this point (the graphics field, picture space, and viewport) specify *where* on the graphics *page* the picture is drawn (or plotted). The graphics window specifies where in the viewport the picture is shown, but it can also determine what the picture looks like: its shape, ratio of one side to the other, size, how much of the picture is shown, and so forth. All these characteristics are controlled by the current definition of the graphics window. The graphics window must be defined *after* the graphics hierarchy routines for the page and field are specified; if you define a graphics window, and then create a page or define a field, the default graphics window is used (x and y = 0 through 100).

Defining a Graphics Window

GSWIN – Define a graphics window. GSWIN specifies coordinates that correspond to the boundaries of the viewport. If the coordinates specified by the graphics window have a range less than that used by primitives in the program, only part of the picture will be shown (for this condition, *clipping* must be set on; clipping is described on page 3-55).

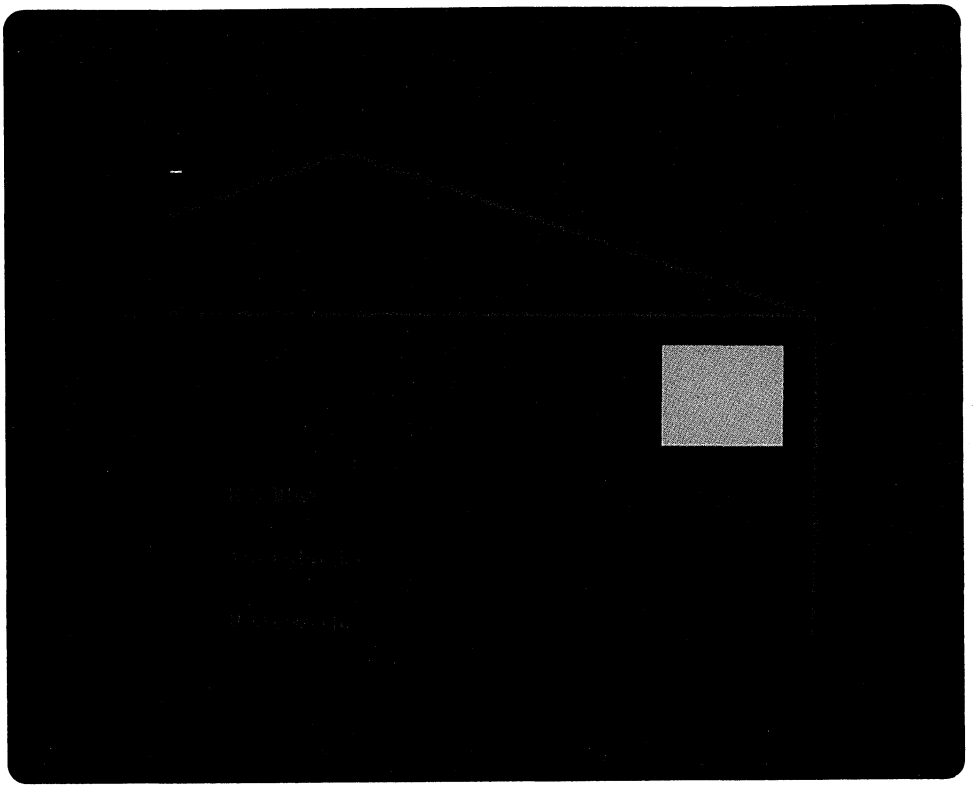
The GSWIN routine uses four parameters: two to specify the beginning and end of the x-range and two to specify the beginning and end of the y-range.

The default graphics window is equivalent to GSWIN,0,100,0,100. However, if all of the other control routines are allowed to default for the 5292 Model 2 (which results in a full-screen page, field, picture, space, and viewport, of normally 24 rows by 80 columns), then the length of 100 units in the x-range is 1.89 times the length of the y-range. If you use GSARC to draw a complete circle, the circle looks like an ellipse with a major axis 1.89 times the length of the minor axis. The ratio of one range to the other is called the *aspect ratio*. To avoid an aspect ratio that distorts graphics (as described above) while still using the default picture controls, you should set the picture space to 1:1 (GSPS,1,1).



Reducing the picture. The graphics window used for the picture can reduce the size of the picture. (The page, graphics field, picture space and viewport used here are the defaults.)

```
CALL GDDM ('GSWIN',-100.0,200.0,-100.0,200.0)
! Set range of graphics window such that picture coordinate range of
! 0 through 100 for x and y are one third the range of
! the viewport
```



Expanding the picture. The graphics window used for the picture can expand the size of the picture. (The page, graphics field, picture space and viewport used here are the defaults.)

```
CALL GDDM ('GSWIN',25.0,100.0,25.0,100.0)
! Set range of graphics window such that picture coordinate range of
! 0 through 100 for x and y are 25% greater than the range of
! the viewport
```

The graphics window coordinate system can be redefined at any point in the program, unless an area-fill has not finished (GSENDA) or the current graphics segment has not been closed. Graphics segments are discussed on page 3-58.

GSQWIN – Query the graphics window. GSQWIN returns the values of the coordinates for the boundaries of the current graphics window.

Clipping

Graphics clipping allows you to use coordinates for routines that are outside the graphics window (the coordinate range that corresponds to the viewport boundary). When clipping is enabled (set on), your program can draw parts of the picture outside the graphics window; the picture is *clipped* at the graphics window (the viewport boundaries). By using clipping with a small graphics window in a program that uses a large coordinate system, you can enlarge a selected portion of the picture.

The clipping state (whether clipping is on or off) is valid for the current page; if a new page is created or another page is selected, clipping will be disabled (set off).

Note: It is not possible to enlarge a graphics image defined by routine GSIMG, because the graphics image is defined in terms of *pixels*, not coordinates. To enlarge a graphics image, you must use the GSIMGS routine described in

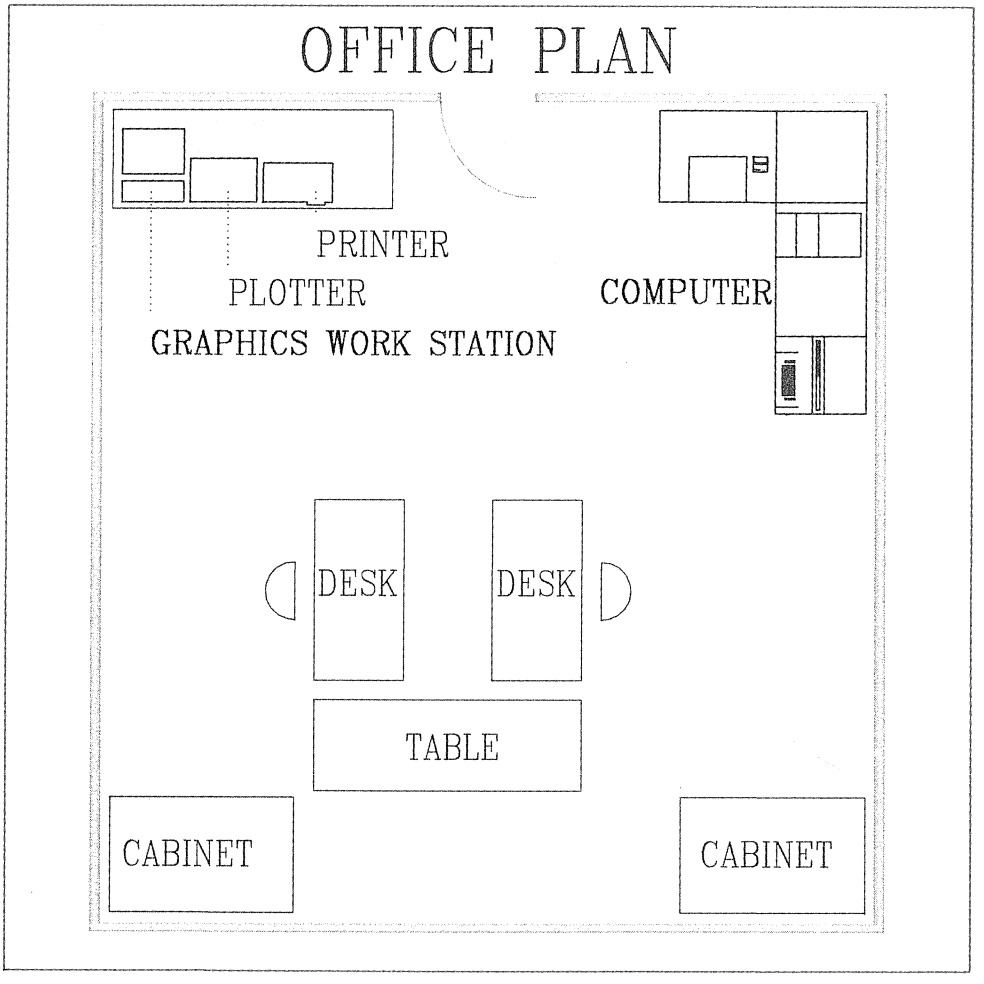
"Drawing Graphics Images" on page 3-35. Also, it is not possible to use clipping with charts defined by Presentation Graphics routines.

Setting the Clipping State

GSCLP – Enable/disable clipping. GSCLP sets clipping on or off.

GSQCLP – Query the clipping state. GSQCLP returns the status of clipping for the current page.

For an example of clipping, imagine a program that draws the floor plan of a room. The program draws the floor plan using a coordinate system of 0 through 1000 in both the x and y directions. To draw the entire floor plan, the graphics window is set to match the coordinates used in the routines.

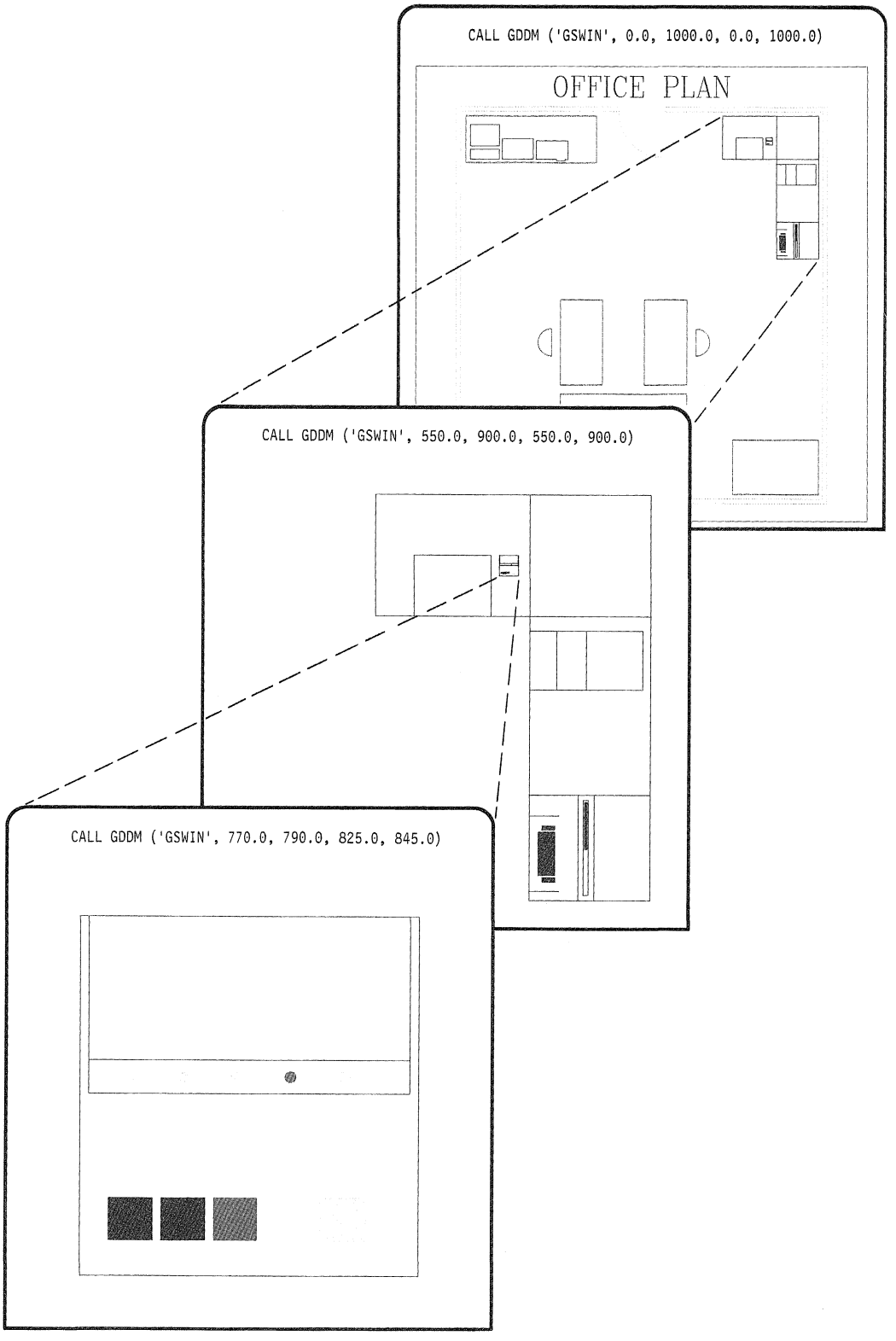


The office. The picture is drawn with aspect ratio set to 1:1 ('GSPS', 1.0, 1.0), and a graphics window that matches the smallest and largest coordinates used in primitives. Clipping can be set on, but doesn't have to be.

```
CALL GDDM ('GSWIN', 0.0, 1000.0, 0.0, 1000.0)
```

With clipping enabled, the graphics window can be changed so that part of the picture can be enlarged in the viewport.

The office. The graphics window can be changed to enlarge a portion of the picture (clipping must be set on).



The Graphics Segment

The graphics segment groups logically-related primitives and their attributes.

Like pages, graphics segments are assigned numbers, and can be created and deleted. Unlike pages, they cannot be changed once they have been created and closed.

When a graphics segment is created, each attribute for primitives is set to its default value.

Graphics segments can be useful when you want part of the picture to be erased at some point after being drawn, or when you want only part of the picture to be updated. You can put your entire picture into one graphics segment, or you can put parts of your picture into several of them.

Graphics segments are associated with a specific page. They can be opened, closed, deleted, or queried.

Creating a Graphics Segment

GSSEG – Create a graphics segment. GSSEG opens a graphics segment with the specified identifier. Identifier 32767 is reserved and cannot be used.

When a graphics segment is opened, all attributes are given default values; opening a new graphics segment might be easier than resetting each individual attribute. The graphics window and viewport cannot be changed while the graphics segment is current.

Note: You cannot explicitly open a graphics segment when you are drawing a Presentation Graphics chart in a picture space; Presentation Graphics manages them implicitly.

Closing a Graphics Segment

GSSCLS – Close a graphics segment. GSSCLS closes the current graphics segment. The current graphics segment must be closed before a new one is opened; once closed, it cannot be reopened.

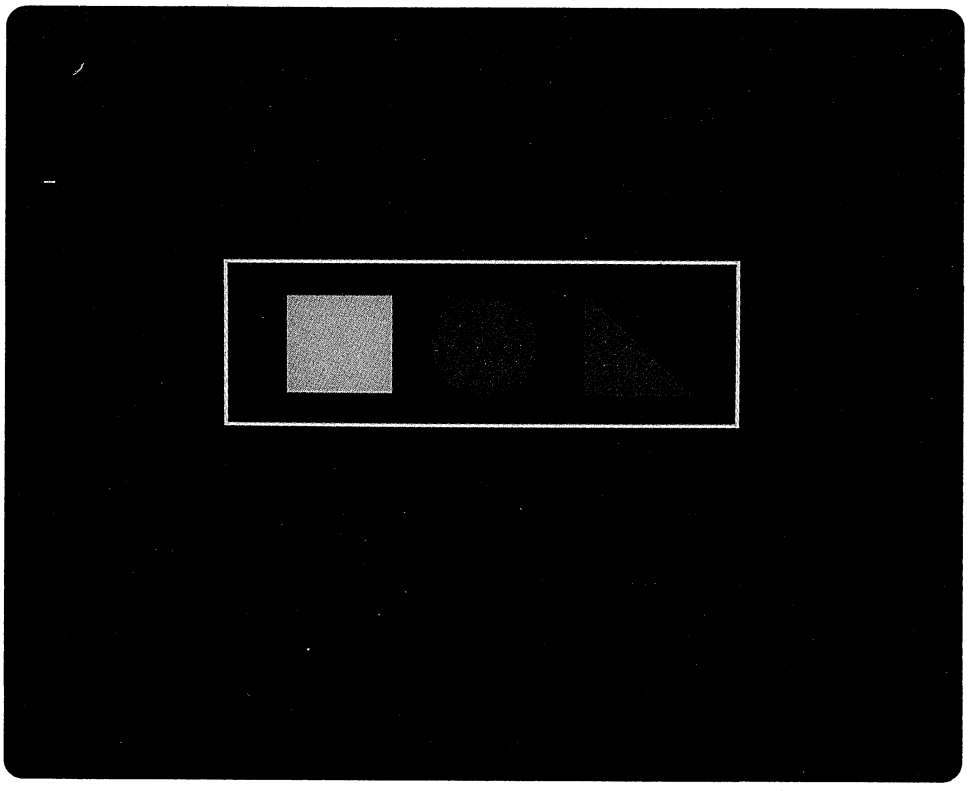
Deleting a Graphics Segment

GSSDEL – Delete a graphics segment. GSSDEL deletes the identified graphics segment. When the graphics segment is deleted, the parts of the picture that were drawn in the graphics segment are erased from the picture the next time an ASREAD or FSFRCE routine sends the picture to a device.

Querying the Number of Graphics Segments

GSQMAX – Query the number of graphics segments. GSQMAX returns the values of the number of graphics segments that have been opened or closed for the current page, and the highest number used for a graphics segment identifier on the current page. You can use this routine to find the highest number used so far and then use GSSEG to create another graphics segment with an unused number.

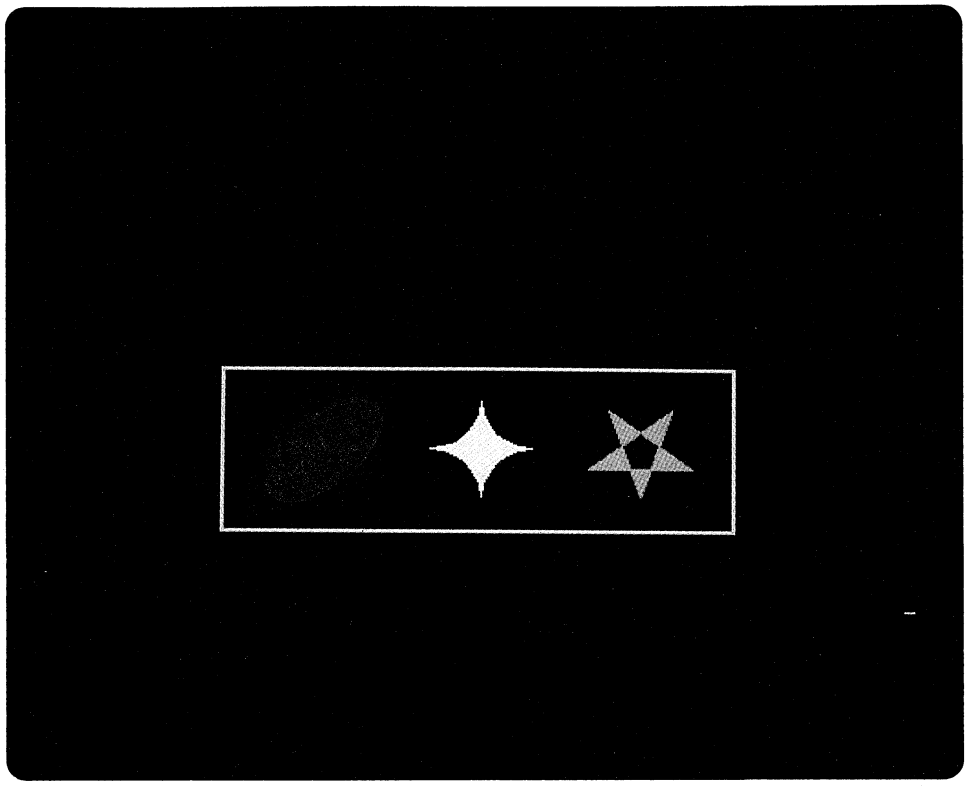
In the next example, assume that a program draws three objects: a square, a circle, and a triangle. All three are associated with the same page, and each object is in its own graphics segment.



Page 1. Three graphics segments are defined.

```
CALL GDDM ('FSPCRT',1,0,0,0)
    ! Create page 1, using default depth, width, type
CALL GDDM ('GSSEG',1)
    ! Create graphics segment 1 for the picture of the square
    .
 (GDDM sets attributes for and draws first object, the square)
    .
    .
CALL GDDM ('GSSCLS')
    ! Close the current graphics segment (number 1)
CALL GDDM ('GSSEG',2)
    ! Create graphics segment 2, (resets all attributes for primitives)
    .
 (GDDM sets attributes for and draws second object, circle)
    .
    .
CALL GDDM ('GSSCLS')
    ! Close the current graphics segment (number 2)
CALL GDDM ('GSSEG',3)
    ! Create graphics segment 3, (resets all attributes for primitives)
    .
 (GDDM sets attributes for and draws third object, triangle)
    .
    .
CALL GDDM ('GSSCLS')
    ! Close the current graphics segment (number 3)
CALL GDDM ('ASREAD',ATTYPE,ATTMOD,COUNT)
    ! Sends picture containing square, circle, and triangle
    ! to the display
```

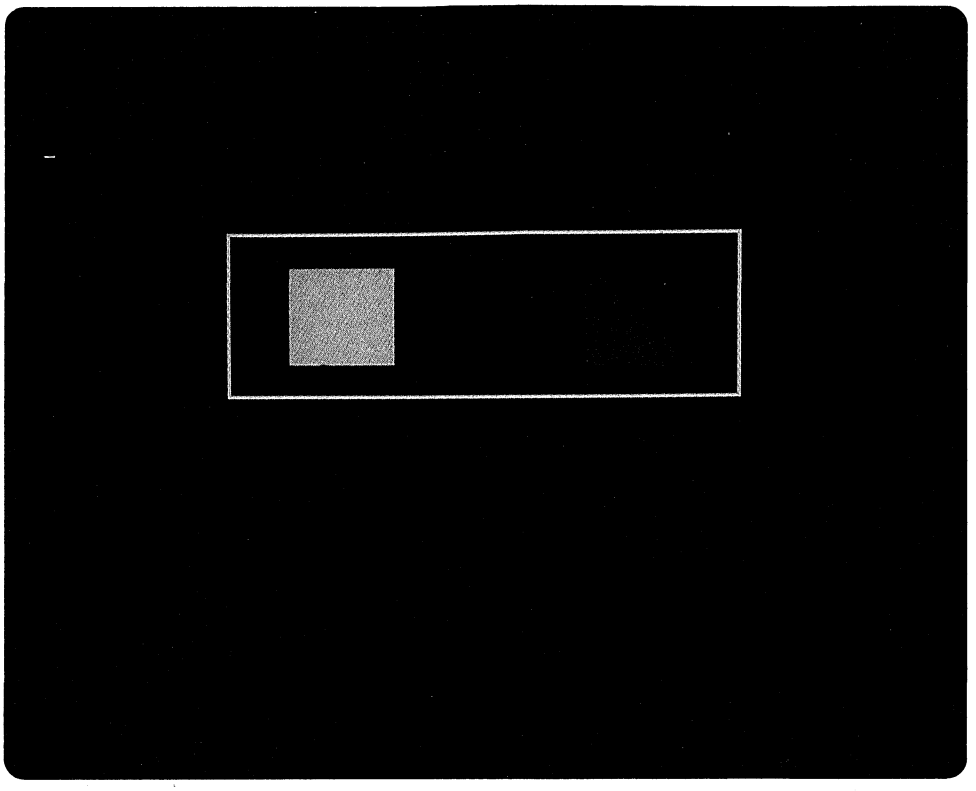
After each graphics segment has constructed its part of the picture, ASREAD sends the picture to the display. ASREAD is followed in the program by a different page that constructs a different picture and sends it to the display:



Page 2. Page 2 is created, and the picture is sent to the display.

```
    .
    .
CALL GDDM ('FSPCRT',2,0,0,0)
    ! Create page 2, using default depth, width, type
    .
    .
 (GDDM draws picture associated with page 2)
    .
    .
CALL GDDM ('ASREAD',ATTYPE,ATTMOD,COUNT)
    ! Sends page 2 picture to the display
    .
    .
```

Now, the program again selects the first page that contains the graphics segments that drew the square, circle, and triangle, but it *deletes* graphics segment 2.



Page 1. The program selects page 1, and shows only graphics segments 1 and 3.

```
    .
    .
CALL GDDM ('FSPSEL',1)
    ! Select page 1
CALL GDDM ('GSSDEL',2)
    ! Delete graphics segment 2
CALL GDDM ('ASREAD',ATTYPE,ATTMOD,COUNT)
    ! Sends page 1 picture to the display
    .
    .
```

GSSDEL deletes graphics segment 2, so when ASREAD sends the picture to the display device, only the square and the triangle appear.

If later in the program, page 1 is selected again and sent to the display device, the circle will still be missing because its graphics segment was deleted and no longer exists in page 1.

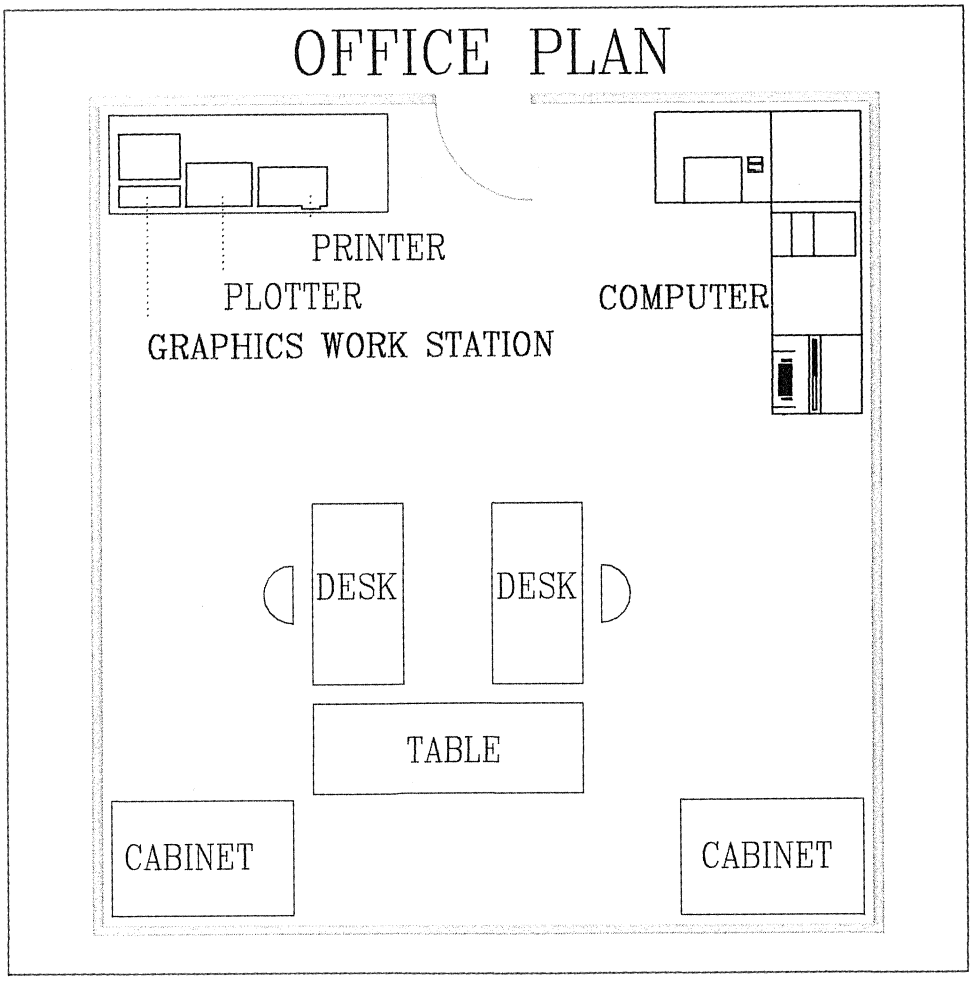
Retained and Temporary Data

Graphics data associated with a graphics segment is called *retained* data, even though graphics segments cannot be reopened like a page can. When you do not open a graphics segment in your program, all elements of a picture constructed by GDDM routines are considered to be *temporary* data. Temporary data differs from retained data (data associated with a graphics segment) in that it is lost after an ASREAD sends that page to the display, or when one of the following routines is encountered in the program:

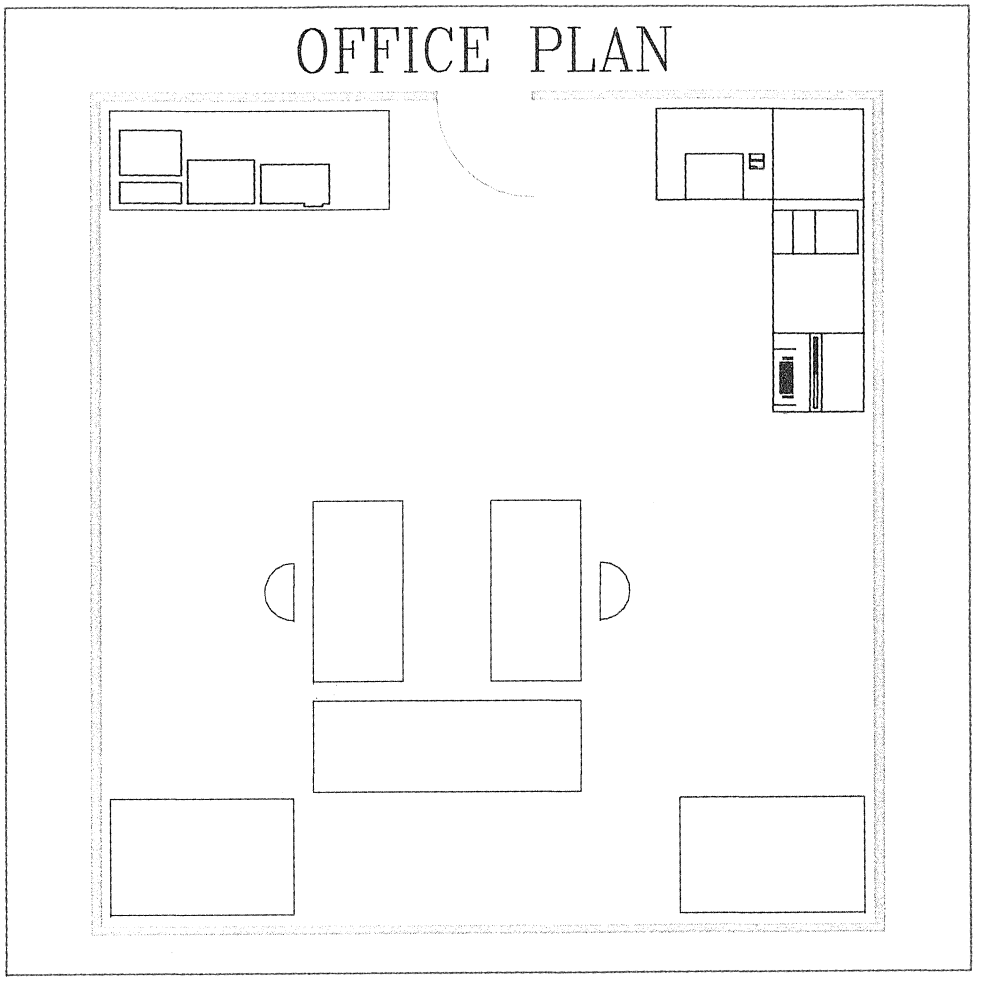
FSFRCE
FSREST
FSPCRT
FSPSEL
GSCLR

Also, temporary data cannot be saved with a *graphics data format* (GDF) file. GDF files are described on page 3-67.

If the program that drew the office floor plan was written so that all parts of the picture were associated with a graphics segment except for the labels, the labels would appear as part of the picture only when the graphics page is the current one at the time of the ASREAD. The labels were defined outside a graphics segment, so they are temporary data.



The office. The labels for the items in the picture appear only because the page containing the labels is the current one.



The office.
When the page is selected later, the labels are missing because they were defined outside a graphics segment.

Device Controls

Device controls determine where the picture being constructed in a page is sent. For a program that produces a picture for a graphics work station, all device control routines can be allowed to default. For a program that sends output to other devices, such as a plotter, some device control routines must be specified.

You can also use a *dummy* device. A dummy device can be a work station that cannot show graphics, such as a 5251 or a 5292 Model 1, or a dummy device can be simulated by using blanks in the name-list parameter of the DSOPEN routine (described later). If you allow all device control routines their default values, you can run your programs from these types of devices to debug and test the programs (although you will not see the graphics portion of the resulting picture).

For more information on the devices that can be used with OS/400 Graphics, see Appendix A, "Devices Compatible with the AS/400 System."

Opening and Closing Devices

You can use the following routines to identify and define or to eliminate a device:

DSOPEN – Open a device. DSOPEN makes a device known to the program. A device that is not the default device must be opened by DSOPEN to be identified to the program as a valid destination for program output. To be used by the program, a device that has been opened with DSOPEN must be activated by DSUSE. Only one device can be active (current) at one time. If a device is in use, a DSDROP routine must be used to deactivate that device before DSUSE activates another that has been opened with DSOPEN.

The default device is the work station from which the program was called.

DSCLS – Close a device. DSCLS closes the device and releases all resources defined for it (including storage-displacing items, such as pages, symbol sets, and color tables).

After a device has been closed, it cannot be used again unless it is reopened with DSOPEN. If DSUSE is specified with the identifier of the closed device, an error occurs. To suspend the use of a device temporarily, use the DSDROP routine.

DSQUID – Query a unique device identifier. DSQUID returns the value of the next available unused device identifier.

DSQUID can be specified to return the value of the next unused device identifier, and this value can then be passed to the DSOPEN routine to avoid conflicts with other device identifiers.

DSRNIT – Reinitialize a device. DSRNIT reinitializes a device to the status it had after it was opened. DSRNIT is equivalent to specifying a DSCLS followed by a DSOPEN.

Using Devices

To be used by the program, the previously-opened device must be activated by DSUSE. If a device is in use (except for the default device), a DSDROP routine must be used to deactivate it before DSUSE activates another. Only one device can be active at a time.

The following routines control the device usage:

DSUSE – Specify device usage. DSUSE activates the device for usage. The device must have already been opened by DSOPEN.

DSDROP – Discontinue device usage. DSDROP specifies that the device is no longer the current device. None of the resources for the device are released (pages and symbol sets). The device can be restored to use by a DSUSE routine.

DSQUSE – Query device usage. DSQUSE returns the identifier of the current device. Additional information about the current device can be returned by the FSQDEV routine (described in the next section).

Querying the Device Characteristics

The following routines can be used in a program to retrieve device information:

DSQDEV – Query specific device characteristics. DSQDEV returns information about the specified device. The information is the same as that specified when the device was opened, or is information about the default device.

FSQDEV – Query device characteristics. FSQDEV returns information about the current device. The device identifier of the current device can be returned by the DSQUSE routine.

Sounding the Device Alarm

You can specify that the program sounds the alarm on the current device (it must be a graphics work station) when the current page is sent to the screen for display.

FSALRM – Sound the device alarm. FSALRM sounds the device alarm when an ASREAD or FSFRCE routine is encountered. The alarm can be used to alert the user when the current page is displayed.

Controlling Graphics: Summary

The part of this chapter you have just finished reading (the second section of Chapter 3, "Using GDDM") showed you the GDDM routines you can use to control:

The program, including the graphics environment and error-handling

Displaying the picture

The characteristics of the items in the graphics hierarchy

The devices used by the graphics program.

The program must initialize the graphics environment before any graphics routines can be called. You can add statements in your program to control the handling of errors that occur in the program. In your program you can control the point at which the current page is sent to the current device.

You can use the items in the graphics hierarchy to control the placement and look of the picture when it is sent to the device. The page is the unit of display that is sent to the device, and its size can be defined. The graphics field further defines the size of the picture on the page, and the picture space sets the ratio of one side of the picture to the other in the graphics field. The viewports define which parts of the picture space are used for pictures, and the graphics window determines (with clipping) how much of the picture is shown in the viewport. Graphics segments group together primitives in the program that can be later deleted from the page.

Device control routines can be used to select which device is the current one. More information about the device control routines can be found in Appendix A, "Devices Compatible with the AS/400 System."

The next section, "Using Graphics Data Format Files," shows you how the graphics data format file can be used to capture an OS/400 Graphics picture. The captured picture can be displayed on the AS/400 System or sent to another system for display, without running the program that generated the picture.

Using Graphics Data Format Files

When your application program uses GDDM routines to draw a picture, an internal graphics data format file is generated. It is this graphics data that the AS/400 System converts to a data stream appropriate for interpretation by the graphics work station (or an associated device), or by a work station printer capable of graphics.

GDDM can capture the graphics data format (GDF) file. The GDF can be:

- Saved in a database file. The GDF can be retrieved from the database file and displayed or plotted at any time. This saves the processing time necessary to generate the picture each time the program is run.
- Sent to another system or device for interpretation. The GDDM program product on the System/370 family of data processing equipment can interpret GDF, and software is available or can be written for interpreting GDF with other systems.
- Produced on the System/370 or System/36 (from BGU) families, or from the AS/400 Business Graphics Utility, interpreted by the AS/400 System and the resulting chart or picture produced on an AS/400 graphics-capable device.

The limitations for using GDF are that the picture held as a GDF cannot be altered or changed (the program must be changed and executed to generate a new GDF), and that the transition from program output to GDF to picture can sometimes change the look of the picture (its aspect ratio, for example), and characters generated by a display file cannot be included in the GDF; only the graphics portion of the picture is saved.

Before you retrieve the GDF, you must specify that the output of the program will be sent to a dummy device (for more information about dummy device support, see "Non-Graphics Devices" on page A-11).

Retrieving Graphics Data

GSGETS – Start retrieving graphics data. GSGETS identifies the graphics segments of the current page whose graphics data will be captured.

GSGET – Retrieve graphics data. GSGET puts the graphics data into a variable in your program. The GDF can then be written by the program to a database file.

GSGETE – End retrieving graphics data. GSGETE ends the retrieval operation started by GSGETS. For a program that retrieves the GDF for one page and then terminates, GSGETE can be omitted.

Drawing a Picture with a Graphics Data Format File

GSPUT – Draw data from a graphics data format file. GSPUT converts the GDF back to a picture that can be displayed or plotted.

The GDF routines work together. After the GDDM (or Presentation Graphics) routines have been processed for a program using dummy device support, the GSGETS routine identifies the graphics segments of the current page whose graphics data will be captured. Presentation Graphics manages its own graphics segments, so GSGETS should specify that they all are to be retrieved.

GSGET retrieves the graphics data. If the buffer (the program variable) is large enough to contain the entire GDF, the length value of the GDF is returned in a

parameter of GSGETS. If the buffer is not large enough, you can use GSGET in a loop that writes the graphics data from the buffer to a database file and checks the data length each time through the loop. When the data length is zero, the loop can end. For example:

```
00010 CALL GDDM ('FSINIT')           ! Initialize GDDM
00020 OPTION BASE 1                  ! Set subscript base
00030 REM ************ Dummy device routines **********************
00040 INTEGER PLST                   ! Declare integer
00050 DIM PLST(1) : PLST(1) = 0      ! Dimension, assign value
00060 DIM NLST$(1) : NLST$(1) = ' '  ! Dimension, assign value
00070 CALL GDDM ('DSOPEN',2,1,'5292M2  ',0,PLST(),1,NLST$(1))
00080 ! Open device 2 of family 1 named 5292M2, using no processing
00090 ! options in the PLST and dummy device name ' ' in the NLST$
00100 CALL GDDM ('DSUSE',1,2)        ! Use device 2 as current dev
00110 REM ************ Open file and graphics segment ******************
00120 OPEN #10: "NAME=GDF,LIB=YOURLIB,FORMAT", OUTPUT  ! Open file
00130 CALL GDDM('GSSEG',1)           ! Open graphics segment (if not PGR)
        .
    (Execute graphics program)
        .
00530 CALL GDDM('GSSCLS')                  ! Close graphics segment
00540 REM ************ Retrieve graphics data ********************
00550 INTEGER GDFL, BUFLENG                ! Declare integers
00560 DIM ARRAY(1): INTEGER ARRAY          ! Declare array
00570 DIM BUFF$*255                        ! Dimension char variable
00580 BUFLENG = 255                        ! Set buffer length
00590 ARRAY(1) = 0                         ! Set element 1
00600 CALL GDDM('GSGETS',1,ARRAY())        ! Start GDF retrieve
00610 LOOP:  CALL GDDM('GSGET',BUFLENG,BUFF$,GDFL)
00620   IF GDFL > 0 THEN WRITE #10, USING 630: BUFF$ ELSE GOTO 650
00630   FORM C 255
00640 GOTO LOOP
00650 CALL GDDM ('FSTERM')                 ! Terminate GDDM
00660 STOP                                 ! Stop execution
00670 END                                  ! End BASIC program
```

BASIC programs are limited to character variables of a maximum 255 bytes. Because the input buffer is a character variable, the GDF has a record length of 255. Therefore, the file being used to hold the GDF should have a record length of 255 also.

GSGETE ends the capture of the graphics data. In a program that captures the graphics data, no ASREAD routine is necessary. There is no GSGETE in this program as only one page is being retrieved (the default page, graphics segment 1).

GSPUT is used in a different program to draw the graphics data saved earlier:

```
00010 CALL GDDM ('FSINIT')              ! Initialize GDDM
00020 OPTION BASE 1
00030 INTEGER ATTYPE,ATMOD,COUNT,CTL,LENG,VSTRING
00040 DIM BUFF$*255
00050 REM ************** Open file *********************************
00060 OPEN #10: "NAME=GDF,LIB=yourlib,FORMAT", INPUT
00070 REM ********** Set picture characteristics **********************
00080 READ #10, USING 100: ORD$,LNG$,TYPE,XLO,XHI,YLO,YHI
00090           ! Read the first record of the GDF
00100 FORM C 1,C 1,B 2,B 2,B 2,B 2,B 2
00110           ! Set format to 1-byte character and 2-byte binary
00120 CALL GDDM('GSWIN',XLO,XHI,YLO,YHI)
00130           ! Set picture space to picture space used in GDF
```

```
00140 XD = ABS(XHI - XLO): YD = ABS(YHI - YLO)
00150  IF XD <= YD THEN CALL GDDM('GSPS',XD/YD,1.0) ELSE CALL GDDM('GS&
&PS',1.0,YD/XD)
00160 REM *********** Put records from GDF ********************
00170 CTL = 2 : LENG = 255
00180 REREAD #10, USING 210: BUFF$     ! Read file again
00190   LOOP:  CALL GDDM('GSPUT',CTL,LENG,BUFF$)    ! Put GDF
00200     READ #10, USING 210: BUFF$ EOF 240        ! until empty
00210     FORM C 255
00220     GOTO LOOP
00230 REM ****** Send picture to display and terminate ****************
00240 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)       ! Display picture
00250 CALL GDDM ('FSTERM')                          ! Terminate
00260 STOP                                          ! Stop execution
00270 END                                           ! End BASIC program
```

The program reads the first record of the GDF and uses the graphics window values found there to set the current graphics window and picture space (the GDF sets its own graphics window using x and y coordinates that can be as high as 32,767).

GSPUT draws the graphics data and when an ASREAD or FSFRCE routine is encountered, the picture is sent to the display device.

More information about the GDF routines and the format of GDF files can be found in the *GDDM Programming Reference* manual. The following example shows a 400-byte record contained within a small GDF file:

```
x...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9
0000007F0052|7000|7000|7000|7000|7000|700000500200002|0F00|400000|2010|10C00C01|801901|70F|44444444444
1A0200FF00B4|2241|2221|2250|2210|2230|0C0001400000006|2F2C0|340000|8080|9114CD9D|14999D|10F|00000000000
     A      B  B  B  B  B      C       D  E  F  G H J  K     L  M N   P
```

A

```
0 0 0 0 0 0 7 F 0 0 5 2
1 A 0 2 0 0 F F 0 0 B 4
```

- High y Value
- Low y Value
- High x Value
- Low x Value
- Type 2 (System /38 or the AS/400 system)
- Length 10 (X'0A')
- Comment order

B Segment attribute orders

C Segment start order

D Color order

E Color mix order

F Set tag order

G Pattern order

H Line type order

J Line width order

K Line order

L Line order at current position

M Segment close order

N FF - End of GDF file

P 40 - Each remaining byte in the GDF is filled with X'40' (blank)

RV2S003-0

Summary of This Chapter

In Chapter 2, "The Application Program Interface to Graphics," the simple program that drew the envelope introduced you to the basic ideas of GDDM: routines are called from application programs written in high-level languages to build a picture. When the picture is built, the program sends it to a device.

Chapter 3, "Using GDDM," showed you more of the capabilities of GDDM routines. Specifically, the topics presented in this chapter were:

Drawing pictures
The first part of the chapter showed the GDDM routines you can use in application programs to define pictures. There are two types of GDDM routines:

Primitives — the basic elements of a picture, such as lines and graphic symbols

Attributes — characteristics that can be assigned to primitives, such as line width or type

Controlling graphics
The second part of the chapter showed the GDDM control routines you can use to manage the graphics program, picture, and the devices it uses.

Program control routines enable you to initialize and end the graphics environment, handle errors, set characteristics for and manage the pictures, and send the program output to a device.

Device control routines allow your program to select devices that receive program output, and to specify characteristics for those devices.

Using graphics data format files
The last part of the chapter showed the use of GDF (graphic data format) file routines in saving the output of a GDDM (or Presentation Graphics) program for later display on the AS/400 System or another system.

At this point you should experiment with GDDM routines in simple programs to gain a better understanding of them, and to get ideas for application programs that use them.

Chapter 6, "Graphics Application Program Examples," shows some examples of complete programs; you can copy those programs and experiment with them, and perhaps use them as a basis for your own programs.

Here are some points to remember:

1. The QGDDM library contains the GDDM routines. If you use a graphics symbol set other than the default set, or if you use the Pascal or PL/I high-level language to write your programs, QGDDM must be in your library list; QGDDM contains the entry point tables for Pascal and PL/I. (The other languages, BASIC, COBOL/400, and RPG/400, use a slightly different type of CALL interface, and do not require the library. For more information on the differences between languages with respect to graphics programming, see Chapter 5, "OS/400 Programming Considerations.")
2. FSINIT must be specified before any calls to other GDDM routines are performed by your program.

3. All the items in the graphics hierarchy can be allowed to default; however, you can specify GSPS(1,1) for a picture space with an aspect ratio of 1:1 to ensure that dimensions in the x range equal to those in the y range. Otherwise, squares will look like rectangles and circles will look like ovals, and so forth.

   If all items in the graphics hierarchy are allowed to default, the range of the graphics window is 0 through 100 in both the x and y ranges, and the viewport is the entire page. Unless clipping is enabled, a routine that positions a primitive outside this range will produce unpredictable results.

4. Data types must be declared according to the high-level language being used.

For more detailed information about each routine, refer to the *GDDM Programming Reference* manual.

Chapter 4. Using Presentation Graphics

The Presentation Graphics program that drew the line chart in Chapter 2, "The Application Program Interface to Graphics," used many of the same routines and program statements as the GDDM envelope program. In fact, the Presentation Graphics program used only one graphics routine that is not described in Chapter 3, "Using GDDM," and that routine is CHPLOT, which is a Presentation Graphics routine used to draw line charts or scatter plots.

Presentation Graphics routines like CHPLOT take data and convert it to a picture. You can use charts to show large amounts of data in a picture that provides an equally large amount of information, but take much less time to comprehend.

Presentation Graphics routines enable you to write programs that convert your real, online data on the AS/400 System into charts. These charts can be used to present an idea to an audience (in hard-copy form), or to make your data easier to understand (in work station display form).

Note: A similar product is the IBM AS/400 Business Graphics Utility licensed program product, program number 5738-DS1. The AS/400 Business Graphics Utility (BGU) offers a menu-driven, interactive method of defining business graphics similar to the graphics constructed by Presentation Graphics routines. For more information, refer to the *BGU User's Guide and Reference* manual.

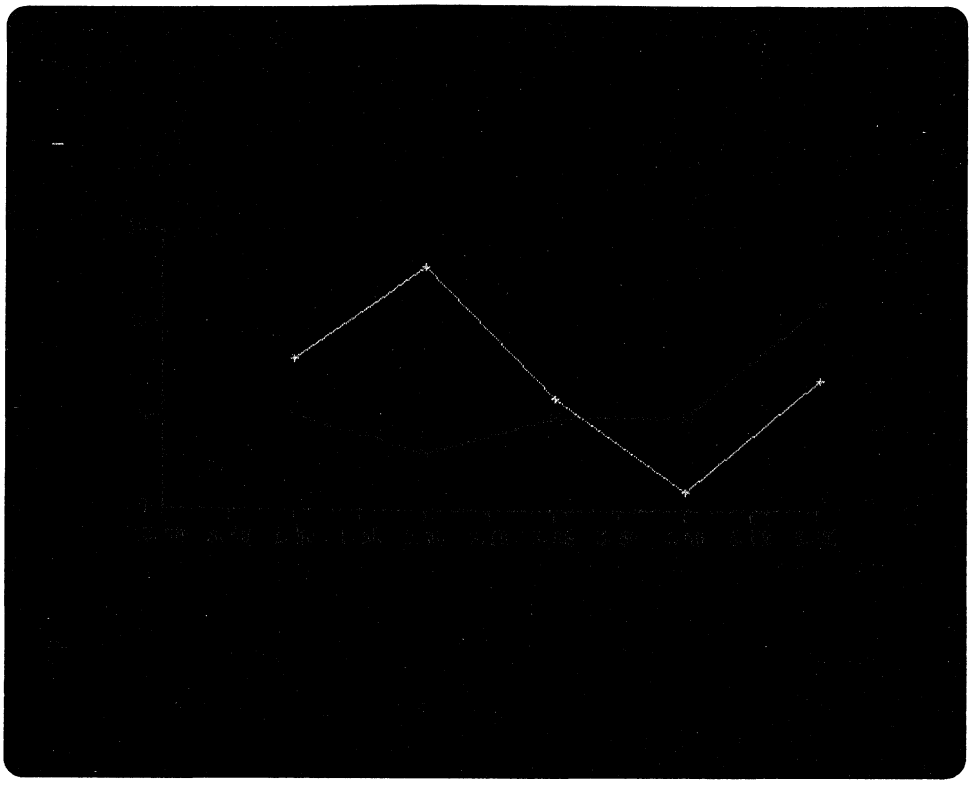
The first part of this chapter introduces the types of chart that you can construct, and shows differences between the types of data and chart format that you can use.

The second part of this chapter shows you the Presentation Graphics routines you can use, in the order they might appear in a typical program.

Understanding Presentation Graphics Routines

The chart shown in "Drawing a Simple Chart with Presentation Graphics Routines" on page 2-11 was a line chart that used two lines to represent data. Each point on the lines showed the relationship of one value (on one axis) to another (on the other). When each point was in position, lines were drawn to connect each point to the next, which resulted in a series of connected lines, considered as one line. Each line represented an entity called a *data group*. Each data group showed a relationship to the other data group, as well as a relationship to each axis. (In charts, each line, bar, pie, and so forth that represents a data group is called a chart *component*.)

© Copyright IBM Corp. 1991



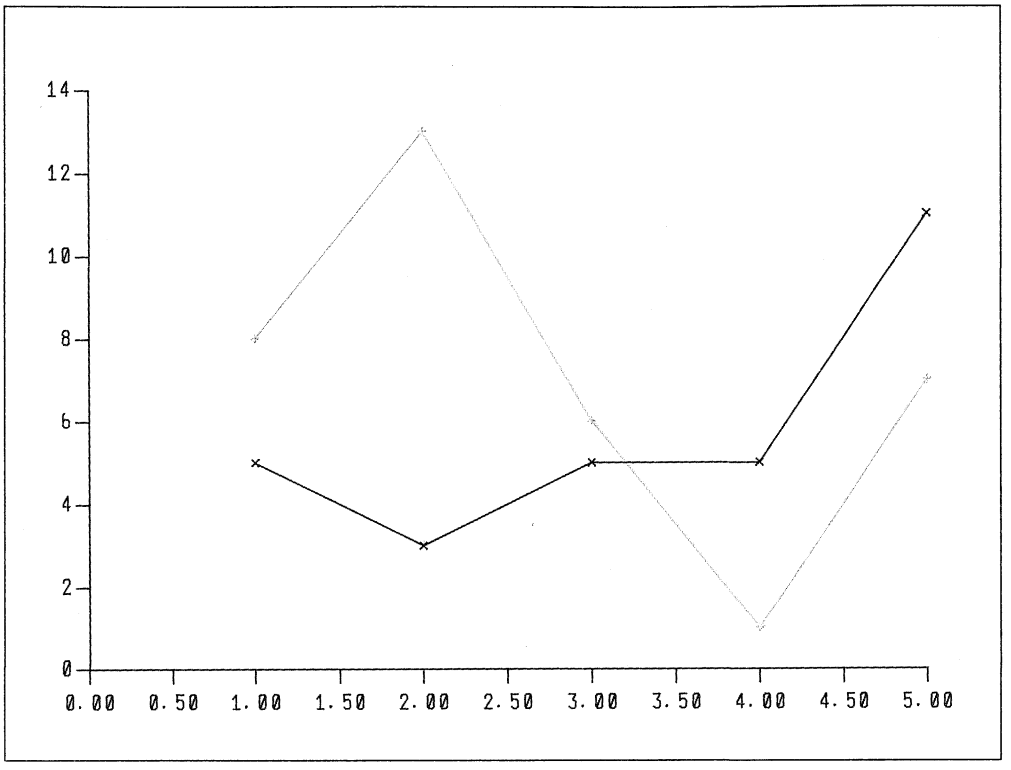
Simple line chart. The line chart was drawn by a BASIC program that used defaults for all Presentation Graphics chart features. This illustration shows the chart as produced on the 5292 Model 2.

The chart condensed the information from the data groups (the data groups were the arrays passed to the CHPLOT routine) into a compact and understandable format. All of the chart formats available through Presentation Graphics can be used to transform data group values into charts. However, different types of data require different types of chart.

The first part of this section shows you chart formats that are available with Presentation Graphics routines, and offers some ideas on the use of each type of chart.

This chapter shows charts produced on the IBM personal computer with work station function (WSF) and the IBM 6180 Plotter. A chart looks different depending on the device it is sent to. For example, when default chart features are used, Presentation Graphics puts more intervals on the axis scale and changes the aspect ratio for the chart area if the chart is shown on the plotter.

This illustration shows the simple line chart on the plotter.



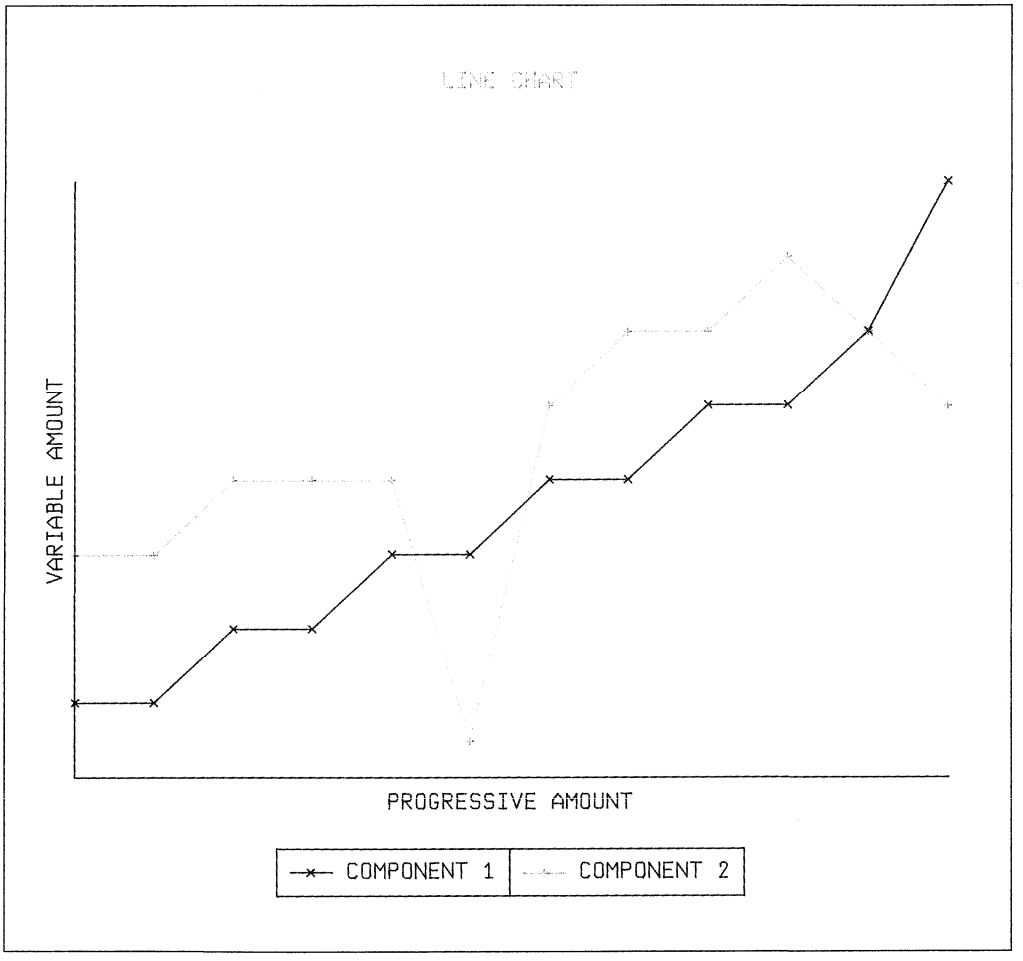
When differences in the chart can result from switching devices, the description of the routine will explain the effect. Refer to the description of CHCGRD on page 4-16 for information on how to avoid these differences.

Chart Types

These are the types of chart available:

Line Charts

Line charts can be used to show change occurring over time. Line charts can represent increases, decreases, trends, and general fluctuations of quantity.



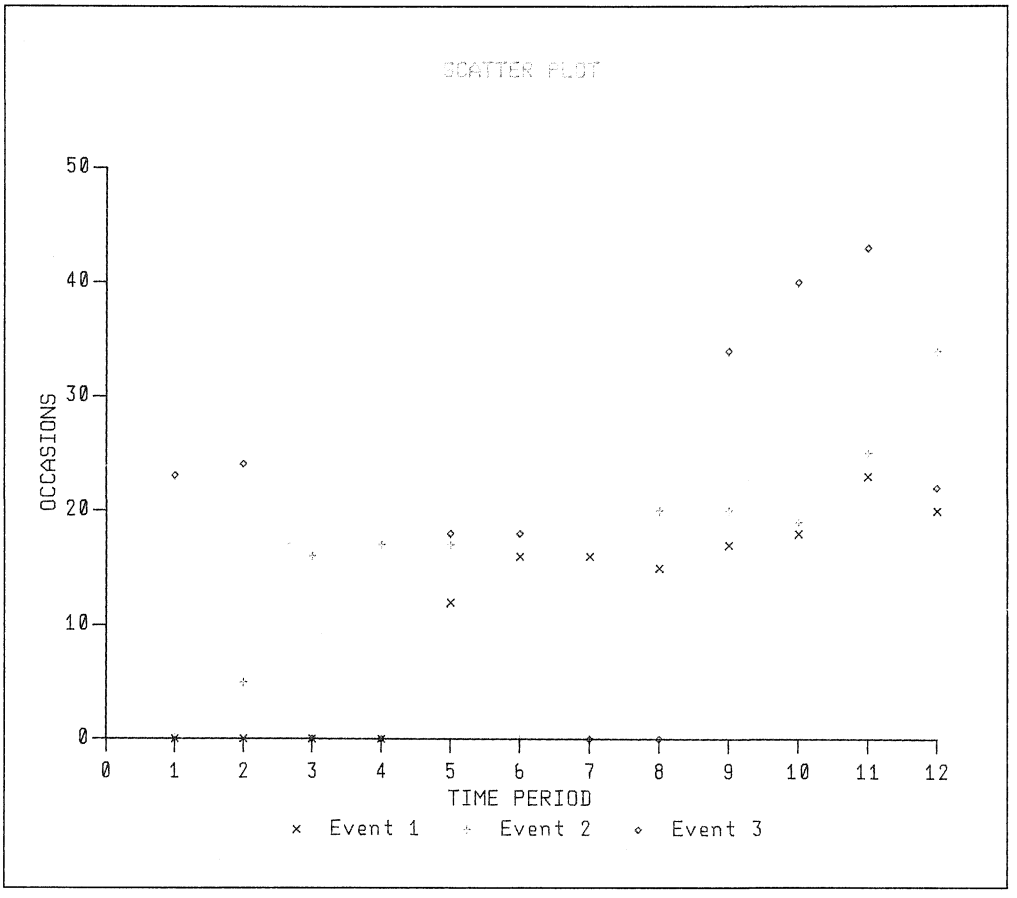
Line chart. The chart represents data as lines.

Each plotted point is shown by a marker; the plotted points are connected to form a continuous line. Each line is assigned a different color. Options exist to convert the sharp corners of the line to a more gentle curve (called *line curving*), or to suppress the display of the lines that connect the points. This is called a *scatter plot* and is described next.

Scatter Plots

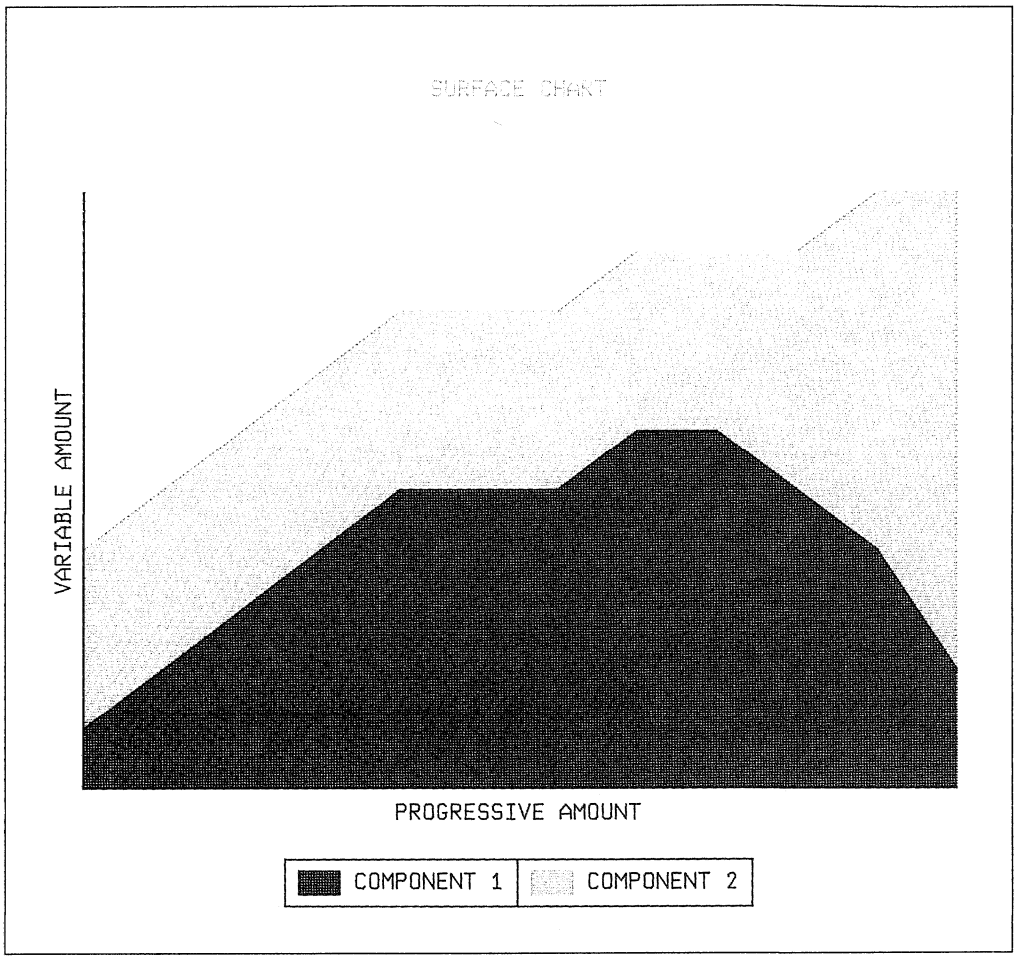
Scatter plots are similar to line charts, except that the lines that connect the data points are not drawn. Scatter plots can be used to show concentrations of data, or the number of occurrences of an event over a period of time.

Scatter plot. The chart represents data as markers (similar to those in the GDDM marker table).



Surface Charts

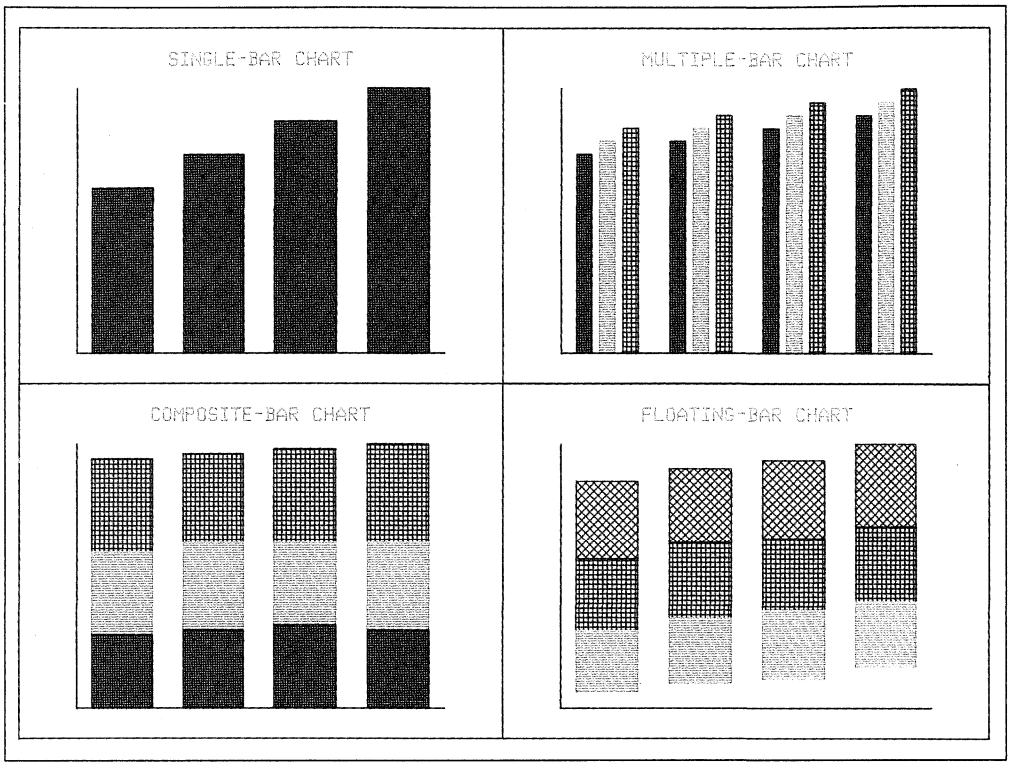
Like line charts, surface charts can be used to show changes occurring over time. Surface charts emphasize volume by shading the area between the lines and the x axis.



Surface chart. The chart represents data as shaded regions.

Bar Charts

Bar chart. Bar charts can be used to show changes occurring over time, parts of an entity, relationships between variables, and comparisons.



Single-bar charts
For showing change over time, single-bar charts can be used when few periods of time are involved. A single-bar chart might be more effective than a line chart for showing comparisons of totals for specific years, if only a few years are being compared.

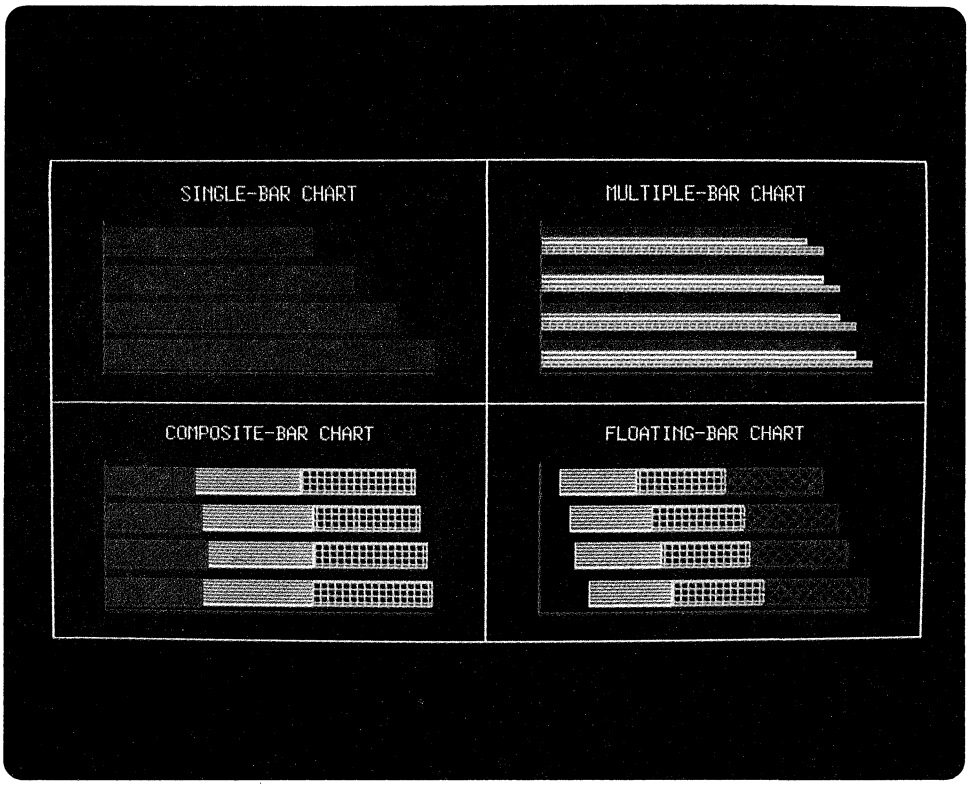
Multiple-bar charts
Multiple-bar charts use slender bars to show relationships of variables for related entities.

Composite-bar charts
Composite-bar charts can be used to show how parts comprise the entity, which is then shown in relation to other entities.

Floating-bar charts
Floating-bar charts are similar to composite-bar charts, except that the first component is not shown. Floating-bar charts can be useful for showing the lower limits of each entity, in addition to the relationship of the elements that comprise the entity.

Horizontal-bar charts

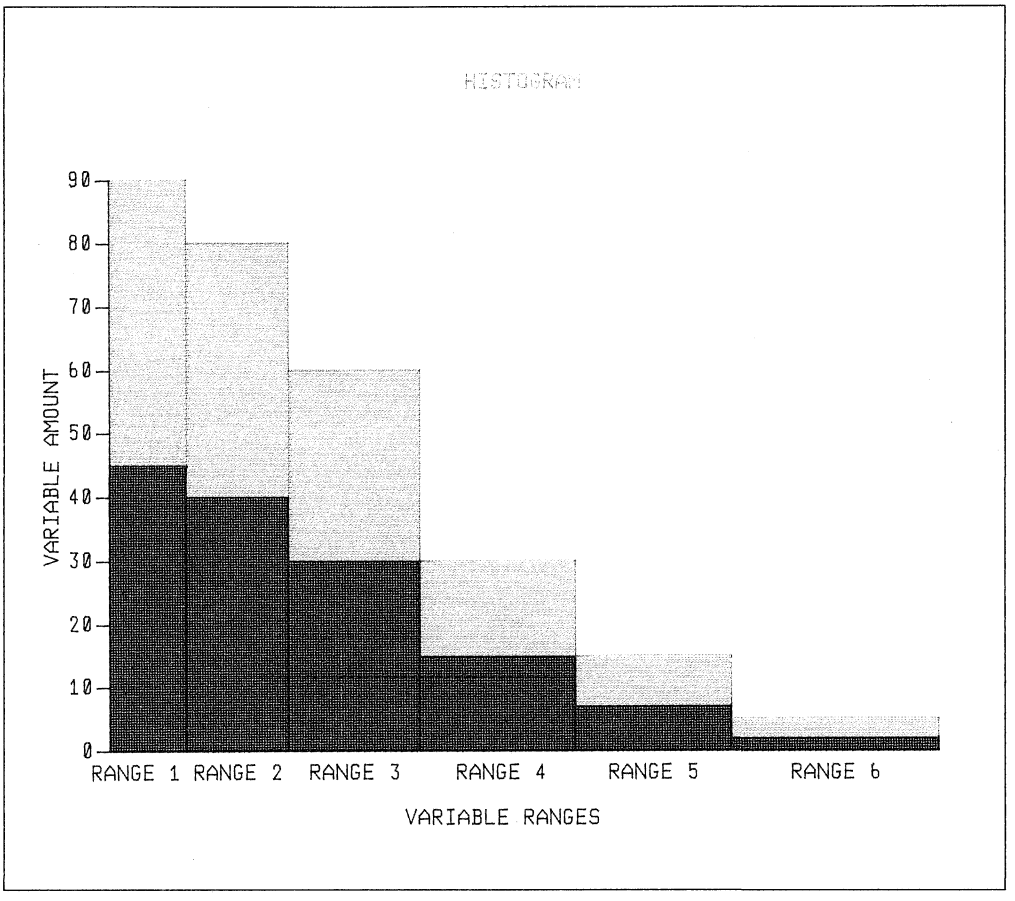
Bar charts usually show the bars rising from the x axis, but you can rotate the chart axis 90 degrees to produce a horizontal bar-chart. A bar chart shown in the horizontal format can place more emphasis on the relationships being illustrated. Any of the four types of bar chart can be shown in horizontal format.



Horizontal-bar chart. A bar chart in horizontal form can be easier to interpret, especially if the bars look tall in vertical-bar format.

Histograms

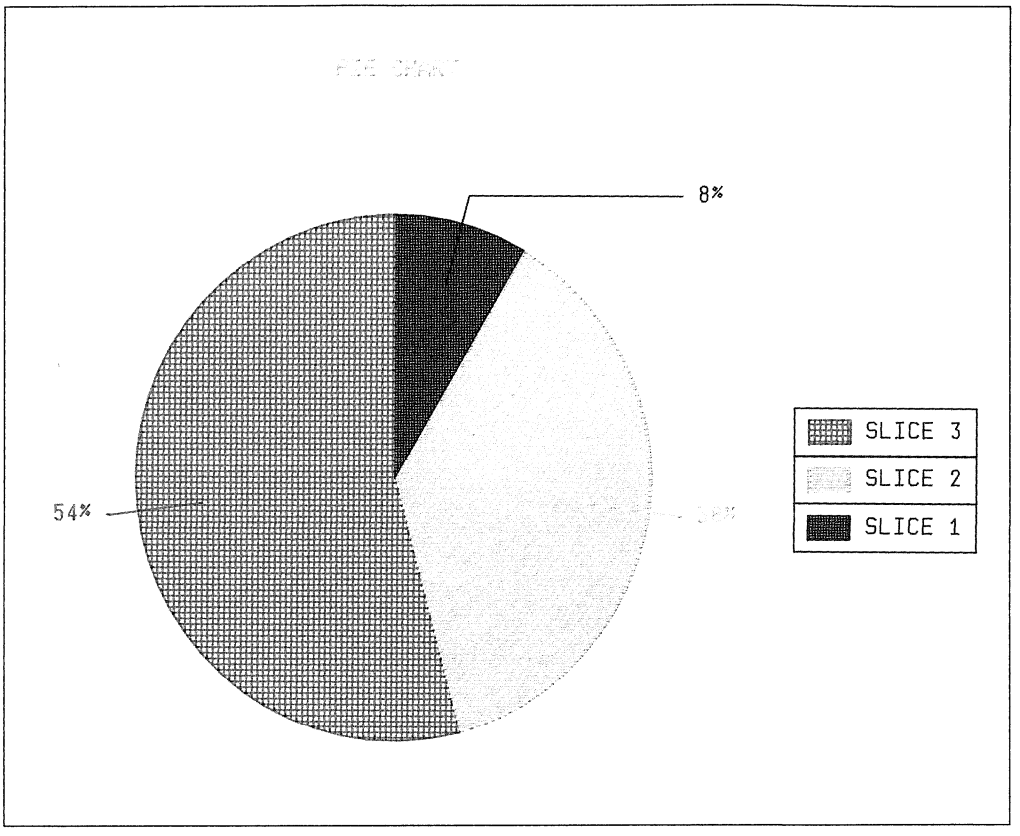
Histograms are similar to bar charts, except that the width of the bar is significant. Each bar represents a variable quantity (relative to the y axis) charted over a range indicated by the width of the bar (on the x axis). If there is more than one component, each is stacked on the first.



Histogram. The chart represents data with both the height of the bar and the width.

Pie Charts

Pie charts are used to indicate the relative size of the elements of an entity, particularly when attention should be drawn to one of the elements. Pie charts are useful for showing percentages.

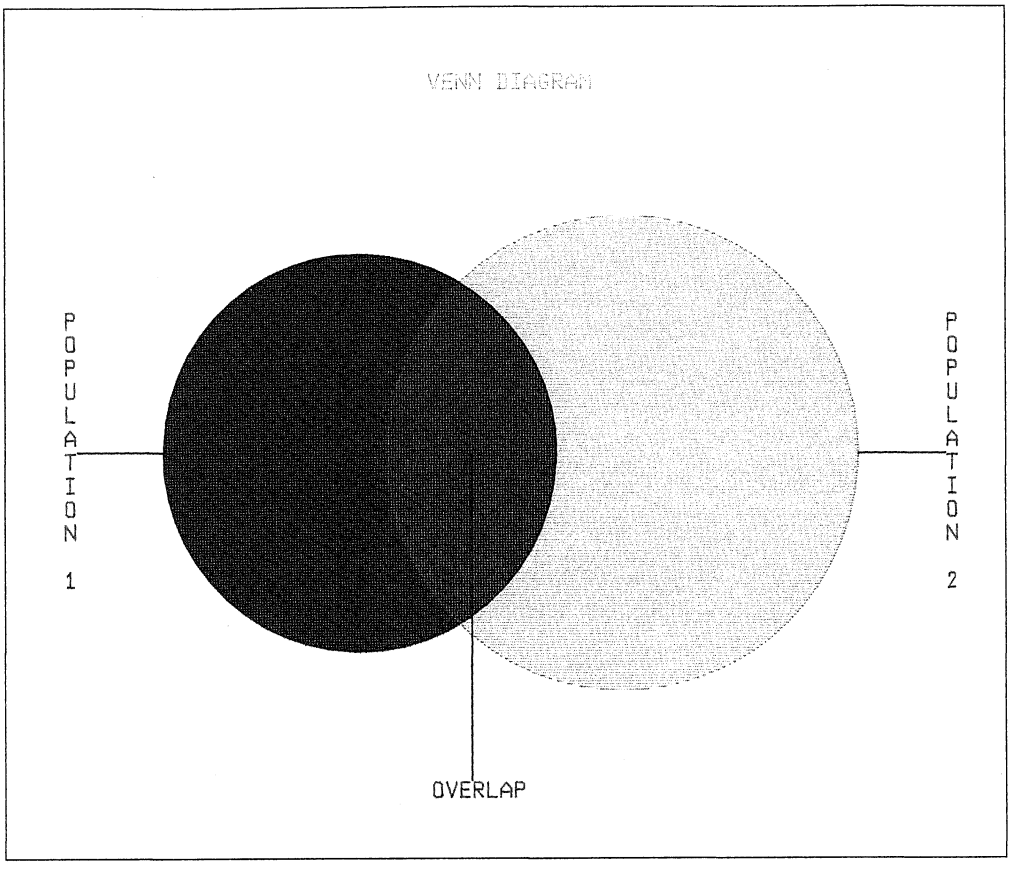


Pie Chart. Data is represented by the size of the pie sectors or *slices*.

You can move one or more slices of a pie chart out from the center of the pie. This is called *exploding* a pie chart.

Venn Diagrams

Venn diagrams are used to show the logical relationship between the overlapping elements of two entities, or populations.



Venn Diagram. Data is represented by the area of each circle and by the area of the overlapping region.

Using Charts to Show Data

The previous section showed you the various chart formats Presentation Graphics offers you. This section gives you some ideas on which chart format is best for your data, and how to improve the appearance and usability of the chart.

Selecting a Chart Type

Some groups of data can be shown with two or three different types of chart format. For example, this table shows information that can be represented by several types of chart:

Year	1984	1985	1986	Total
Heat	375	410	600	1385
Telephone	280	325	410	1015
Utilities	200	270	360	830
Totals	855	1005	1370	3230

1. If you wanted to construct a chart that shows that overall costs increased, you could use a line chart.
2. If you wanted to show that heating costs rose more rapidly than telephone or utility costs, you could use a three-component surface chart.
3. If you wanted to show that heating costs were greater than the other costs, you could use a composite-bar chart (one bar for heating cost and one each for telephone and utility costs).
4. If you wanted to show the percentages of heat, telephone, and utilities that make up the total for each year, you could use three pie charts.
5. If you wanted only to convert all the information from table format to chart format, you could use a multiple-bar chart, one that shows three bars for each year.

You can see from this that many different options are available for charting one set of data.

The most important thing to remember about representing data from tables with charts is this: you should know what information you want to emphasize *before* you select a chart format to use. Try to find a short sentence that describes the information to be shown on your chart:

Sales are up this quarter

The Western Region increased its productivity by 25%

The trend in housing-starts forecasts a good year for builders.

Consider using the descriptive sentence as the main title for your chart. That way, those who interpret the chart have little chance of misunderstanding it.

Drawing Charts with Presentation Graphics Routines

Presentation Graphics routine names are 4- to 6-character mnemonics whose first two characters are CH (short for chart). The rest of the characters in the name represent the function of the routine (CHNOTE, for example, is a routine that writes a note).

Some Presentation Graphics routines specify attributes for axes. Because two axes exist (the x axis and the y axis), Presentation Graphics uses the third character of these routine names to select which of the two axes the attribute should affect. For example, the Presentation Graphics routine CHXSET specifies characteristics of the x axis while CHYSET specifies them for the y axis.

Most Presentation Graphics routines have parameters that define the function in more detail. Parameters can be constants or they can be variables with values assigned to them. Parameter values can be 4-byte binary integers, short floating-point numbers, or character strings.

The Structure of Presentation Graphics Programs

Presentation Graphics programs use a sequence of operations to construct charts:

1. Control operations
2. Chart definition
3. Chart drawing
4. More control operations

Control Operations

In Chapter 2, "The Application Program Interface to Graphics," you learned that for both the GDDM envelope program and the Presentation Graphics chart, the GDDM routine FSINIT was used to initialize the graphics environment. All Presentation Graphics programs must begin with a call to FSINIT.

You can specify other GDDM control routines for your Presentation Graphics program, or you can take the default values. You could, for example:

Specify an error-handling subroutine with FSEXIT

Define a specific page with FSPCRT

Manage symbol sets for use by the program

Manage tables for colors, line types, markers, and shading patterns for use by the program

Open, select, and close devices; the device control routines were discussed in "Device Controls" on page 3-64.

Some GDDM control routines that you *cannot* use are:

Picture spaces (GSPS)
Graphics segments (GSSEG)
Viewports (GSVIEW; for Presentation Graphics, use CHAREA routine)
Graphics windows (GSWIN).

Presentation Graphics defines these items implicitly in the graphics hierarchy.

If you need to return to the GDDM environment to add GDDM-described features to the Presentation Graphics described picture, use the CHTERM routine.

For a list of GDDM routines that can be used in the Presentation Graphics environment, refer to the *GDDM Programming Reference* manual.

Chart Definition

Once the initial control routines have been specified, you can use routines that define the appearance of the chart. You can define:

Chart layout, including:

- What size the chart will be and where it will be placed
- What size the margins will be
- Whether the chart will be framed in a box
- What size and spacing the characters will have.

Chart features and their attributes, including:

- The heading and what it will look like
- The axes and what they will look like
- What the components of the chart will look like
- What other things will be added to the chart and what they will look like, such as:
  - A legend
  - Chart notes

Chart Drawing

When you have defined your chart in the program with any of the chart layout and attribute routines, you can specify a single chart-drawing routine that constructs the chart (it is not displayed until an ASREAD or FSFRCE routine sends it to an output device).

The chart-drawing routine you use depends on the type of chart you want.

What You Can Do in a Program and Where

After you have specified the chart control routines, you define the chart with routines that specify the chart layout and attributes. After that, you call a chart drawing routine to draw the chart.

The two parts of the program (chart definition and chart drawing) are called *state-1* and *state-2*.

State-1 is the mode the program is in *before* the chart is constructed, and state-2 is the mode of the program *after* the chart is constructed.

Once the chart is drawn, you can add more features, such as chart notes and additional reference lines. However, some things you might try to add will not have an effect because the chart itself has already been constructed (but you will get a message). For example, there would be no point in specifying an axis range of 0 through 100 after the already-constructed chart used a range of 0 through 50.

More Control Operations

After the chart has been drawn, you can use routines to terminate the program; or, if you want to add more features to the chart or define a new chart and add it to the picture, you can reset the program.

The information that follows describes the functions of the Presentation Graphics routines. Some routines use integer, floating point, and/or character values for their parameters, values that you can assign in your program. With other routines you can choose options for character strings. In the following descriptions, the default option for each routine is shown **in italics**, followed by the other options available.

For a more detailed explanation of each routine, refer to the *GDDM Programming Reference* manual, which shows the syntax of each and the data types that need to be declared for the parameters.

Designing the Chart Layout

The chart layout routines specify the physical arrangement of the areas that frame the chart. These layout routines apply to all chart types.

Setting the Chart Size

CHAREA – Specify chart area. CHAREA defines the size of the chart, in terms of picture space units (see "The Picture Space" on page 3-48). By default, all of the screen is used.

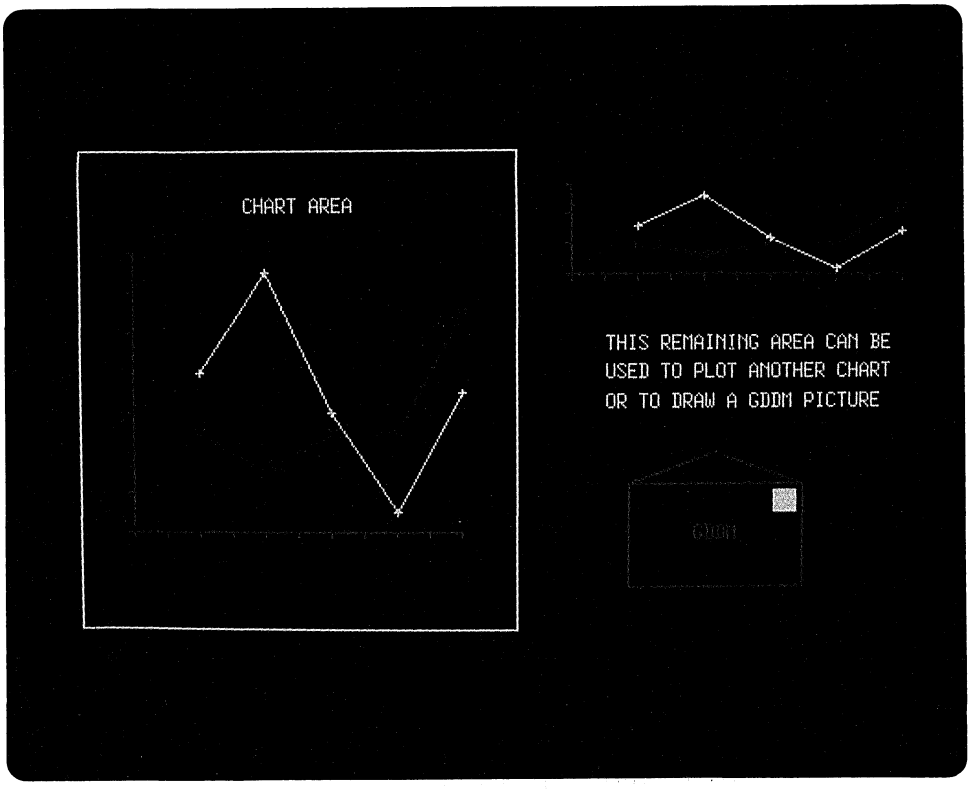


Chart area. The line chart is placed in the chart area defined for the left half of the page.

```
! Define a chart area for the left half of the page
       .
CALL GDDM ('GSQPS', WIDTH, DEPTH)
          ! Query picture space
LET LEFT = 0.0
          ! Left boundary = left margin
LET RIGHT = 0.5 * WIDTH
          ! Right boundary = halfway point
LET BOTTOM = 0.0
          ! Bottom boundary = lower margin
LET TOP = DEPTH
          ! Top boundary = upper margin
CALL GDDM ('CHAREA',LEFT,RIGHT,BOTTOM,TOP)
          ! Sets chart area to left half of screen
       .
       .
  (Other Presentation Graphics routines to define
  and draw a chart in the defined chart area)
```

When you use CHAREA to divide the picture space and then draw a chart, you could reset (CHSTRT) or reinitialize (CHRNIT) Presentation Graphics then use CHAREA to define the other part of the picture space and draw another chart. Alternatively, you could use the CHTERM routine to terminate Presentation Graphics then use GDDM routines to define a viewport and draw a GDDM picture or text there. Note that a viewport (specified by GSVIEW) does for GDDM what CHAREA does for Presentation Graphics. Also, if the GDDM routines precede the Presentation Graphics routines, you must close any opened segments *before* using Presentation Graphics routines.

Setting the Character Size

CHCGRD – Set character spacing/size. CHCGRD specifies the *basic character box size* for all chart text (headings, labels, and notes), and the chart margins (with the CHVMAR and CHHMAR routines described next). CHCGRD specifies this size by giving the number of rows and columns of character boxes needed to fill the entire chart area.

Each routine used to specify attributes for an element of chart text also specifies a multiplier value. The basic character size is multiplied by that value to enlarge or decrease character size. CHCGRD is similar to GDDM routine GSCB, except that the character-box size for Presentation Graphics also determines the size of the chart, the spacing of chart features, and so forth. (GSCB is ignored while Presentation Graphics is initialized.)

By default, the size of the hardware character grid is used. For graphics work stations, the default character grid is 24 rows by 80 columns. In some instances, a chart drawn on a plotter or a printer looks different from the same chart shown on the display, because the default character grid used for the plotter differs from that used for the graphics work station (the default character grid for the plotter varies with the orientation of the paper and the paper size; the default character grid for the printer varies with the row/column dimensions for the printer file being used). To avoid the difference, set the character grid to 24 rows by 80 columns.

The GDDM GSFLD routine can also be used to control the size of the default chart area, or the chart sizes used when two or more charts are defined to fill the entire charting area.

Setting the Chart Margins

CHHMAR – Set horizontal chart margins
CHVMAR – Set vertical chart margins

CHHMAR and CHVMAR specify the size of the horizontal and vertical chart margins. The size is specified in terms of character-size units, based on the character size specified by the CHCGRD routine. When the margin size is increased, the chart size is reduced proportionally.

By default, the horizontal margins are 5 rows of character grid boxes above and below the chart area, and the vertical margins are 10 columns of character grid boxes on either side of the chart area. (The CHCGRD routine overrides the default size of the character grid box. The default character grid is the size of the hardware character cells used on the display device.) If the default right margin of 10 columns is not wide enough for a chart legend, use a CHVMAR routine to widen it.

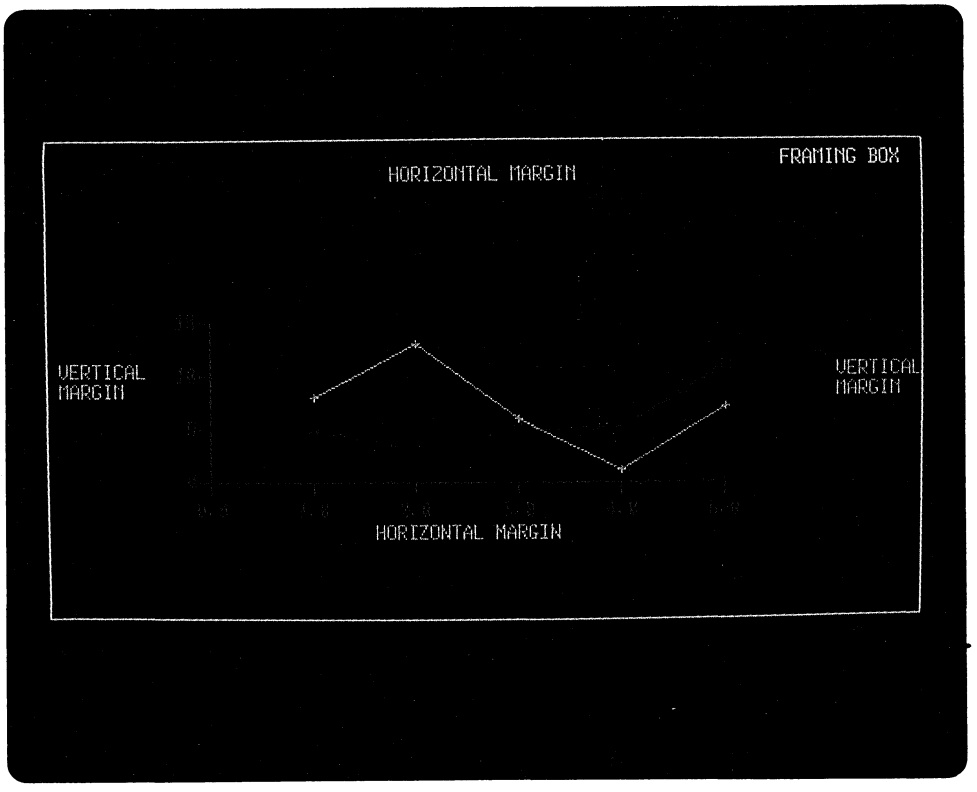


Chart margins. The chart uses horizontal margins of 7 and 9 character boxes and vertical margins of 15 and 18 character boxes. The chart also uses a framing box (described next).

```
CALL GDDM ('CHHMAR',7,9,)       ! Set bottom and top margins
CALL GDDM ('CHVMAR',15,18,)     ! Set left and right margins
CALL GDDM ('CHSET','CBOX')      ! Use chart frame
```

Enclosing the Chart in a Frame

CHSET – Specify chart options. CHSET (**NCBOX**|CBOX|CBACK) specifies whether a framing box is constructed around the chart construction area.

Use the CBOX parameter to draw a framing box around the chart. By default, color 7 is used (white for a graphics work station color table, and the highest pen number for the plotters). Use the CHBATT routine (described next) to set attributes for the framing box.

Use the CBACK parameter to have the entire background shaded with a solid color. By default, the color is the same as that specified for the frame. Use CHBATT (described next) for a different color. CBACK should not be used for Venn diagrams because it suppresses drawing the population circles. If you use CHAREA to set a chart area, the background and the chart frame are sized to fit the chart area and margins.

By default, the framing box and background shading are not drawn.

> **Performance hint**
>
> Specifying a background for your chart is nearly equivalent to a complete page of area-fill. This will substantially increase the time it takes to process and plot your chart.

Setting the Frame Attributes

CHBATT – Set framing box attributes. CHBATT specifies attributes for the framing box produced by CHSET (CBOX) or attributes for the chart background produced by CHSET (CBACK). The attributes are color, line-width and line-type of the frame, and the color of the background.

By default, the frame and background are color 7. The default frame is drawn with a solid, narrow line.

Adding Chart Features

Other Presentation Graphics routines add features that increase the usability of the chart. You can use these routines to control:

- Chart headings
- Reference lines (including axes and axis text)
- Chart legends
- Chart notes

Routines for these chart features (except chart notes) do not have to be specified in any particular order in the program, but they should precede the chart-drawing routine (state-1). Chart notes can be added only in state-2. For information on state-1 and state-2, see page 4-14.

Writing Chart Headings

The chart heading is usually placed at the top of the chart, but it can be placed at the bottom. You can left or right-justify the heading, but it is centered by default. The heading can be more than one line; secondary lines in the heading are also left- or right-justified or centered.

Writing the Chart Heading

CHHEAD – Heading text. CHHEAD specifies the string of characters that is the chart heading. Chart headings can be more than one line long, but cannot exceed 132 characters.

If you want a multiple-line heading, use a semicolon (;) as the *line break* character in the heading. The second line of the heading is then centered below the first line. The semicolon must be counted as a character in the heading even though it does not appear. If you want a semicolon as a character in the heading, use a pair of them (;;). The two semicolons put the single semicolon into the heading, but they do not cause a line break. To put more space between the top of the chart area and the heading, move the heading down by including a semicolon as the first character in the heading string.

No default heading character strings exist. If you use CHHEAD to specify a heading, but do not use CHHATT to select attributes or CHSET to position the heading, the heading is drawn centered at the top of the chart, in standard-size characters of the default color. The CHSET and CHHATT routines are described next.

Suppressing the Chart Heading

CHSET – Specify chart options. CHSET (**HEADING**|NOHEADING) allows or suppresses the heading specified by CHHEAD.

Setting the Heading Attributes

CHHATT – Heading text attributes. CHHATT specifies the color, character mode, symbol set (one loaded by the GSLSS routine), and character-size multiplier used for the heading. If character-mode 2 is specified, the default symbol set is used, and the character-size multiplier serves only to set the spacing of the characters in the heading.

If CHHATT is not specified, the heading is shown with the default color and is written in the default character set with standard-size characters. The default color is number 0 (the same as number 4) for the display and the pen in position 1 for the plotter.

Positioning the Chart Heading

CHSET – Specify chart options. CHSET (**HTOP**|HBOTTOM) positions the heading at the top or bottom of the chart area.

CHSET (**HCENTER**|HLEFT|HRIGHT) positions the heading at the center, left, or right of the chart area.

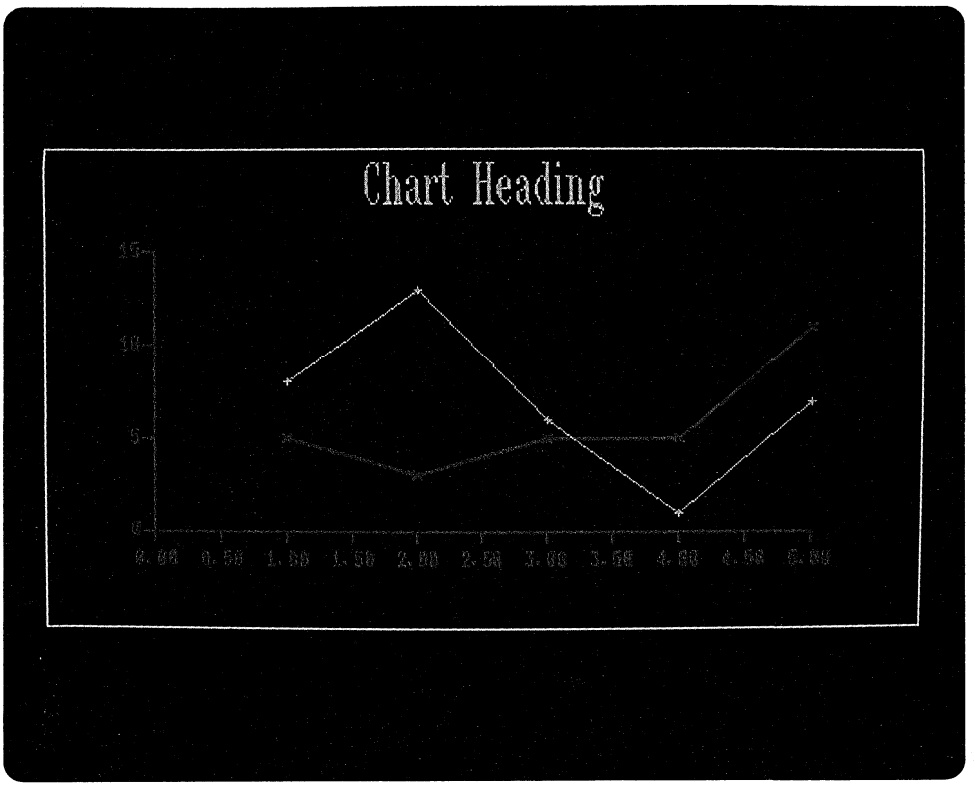


Chart heading. The chart heading should be a descriptive title for the entire chart, one that states the message the chart communicates.

```
CALL GDDM ('GSLSS',2,'ADMUWTRP',66)
! Load vector symbol set ADMUWTRP as symbol set #66
INTEGER HATT                  ! Declare integer
DIM HATT(4) : MAT READ HATT   ! Read attribute array
DATA 2,3,66,300
! 2 = red, 3 = character mode, 66 = symbol set, 300 = size
CALL GDDM ('CHHATT',4,HATT()) ! Set heading attributes
CALL GDDM ('CHHEAD',13,'Chart Heading;(centered)')
! Write 13-character heading with 'character string'
```

Drawing Chart Axes

Most chart types provided by Presentation Graphics use axes (except for pie charts and Venn diagrams). Axes are drawn at 90 degrees to each other; there are usually two primary axes, the x axis and the y axis. Usually, the x axis is horizontal and the y axis is vertical. The axes are the main reference lines for the chart.

Axis lines are lines drawn on the chart to provide a basis for understanding the graphics lines and patterns that represent data. They can be assigned a specific

range of values, which are shown by tick marks (the scale), or numbers, or names of weekdays or months (labels).

The x axis most often represents the *independent variable*, and the y axis represents the *dependent variable*. The independent variable represents the *frame-of-reference* of the chart. Nothing associated with the chart can change the progression of the independent variable. For some chart types, the independent variable is usually *time*: hours, days, weeks, for example. For other chart types, the independent variable represents a related group, such as a group of cargo transporters that has as its elements ships, planes, trucks, and trains.

The dependent variable is the range of data values being used to construct the chart. The dependent variable is the dynamic element of the chart; each value of the dependent variable corresponds to some value in the independent variable. In other words, the dependent variable *depends* upon the independent variable (such as a specific day of the week) for it to have any meaning.

Drawing or Suppressing the Chart Axes

CHXSET – x-axis characteristic
CHYSET – y-axis characteristics

CHXSET or CHYSET (**AXIS**|NOAXIS) specifies whether the axis line is drawn when the chart is constructed. NOAXIS suppresses only the x- or y-axis *line*, not the tick marks, title and labels. You must use CHXSET or CHYSET (PLAIN) to suppress the tick marks for an axis and CHXSET or CHYSET (NOLAB) to suppress the labels for an axis. CHXTTL and CHYTTL are used to specify axis titles; their absence suppresses the axis titles.

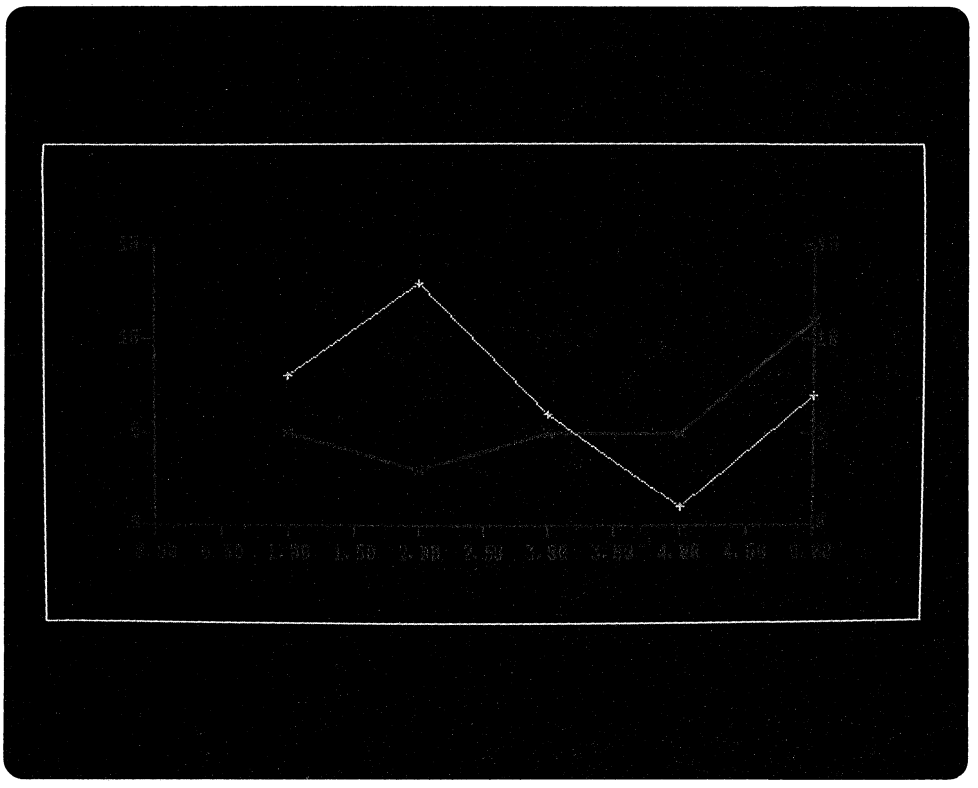
To suppress both axes and their associated tick marks and labels, use the CHSET (NDRAW) parameter (described next). You can use CHDRAX to draw the suppressed axes tick marks, and labels in state-2.

CHSET – Specify chart options. CHSET (**IDRAW**|DRAW|NDRAW) specifies *when* the axes and their associated tick marks and labels are drawn on the chart. You can specify that the axes are drawn before the data components are drawn (the default IDRAW), so that if any part of the component interferes with an axis, the axis is obscured. If necessary, the axes can be drawn each time a component is drawn, so that the axes overlay the components (DRAW), or the axes can be suppressed altogether (NDRAW). If the axes are suppressed, they can be drawn later (in state-2) by the CHDRAX routine (described next).

CHDRAX – Draw axes. CHDRAX draws the axes and the associated tick marks and axis labels. CHDRAX can only be called in state-2 (after the chart drawing routine). CHDRAX is useful when another feature of the chart obscures part of the axis, tick marks, or axis labels.

Setting the Number of Axes

CHSET – Specify chart options. CHSET (**XNODUP**|XDUP|**YNODUP**|YDUP) specifies whether duplicate axes are drawn on the chart. You can specify a duplicate for either the x or the y axis (or both if you use two CHSET routines, one for each axis). The duplicate axis appear at the opposite end of the chart from the primary axis.



Duplicate axes. Duplicate axes are useful for charts where the chart data is difficult to relate to one set of axes. This chart shows a duplicate y axis on the right side.

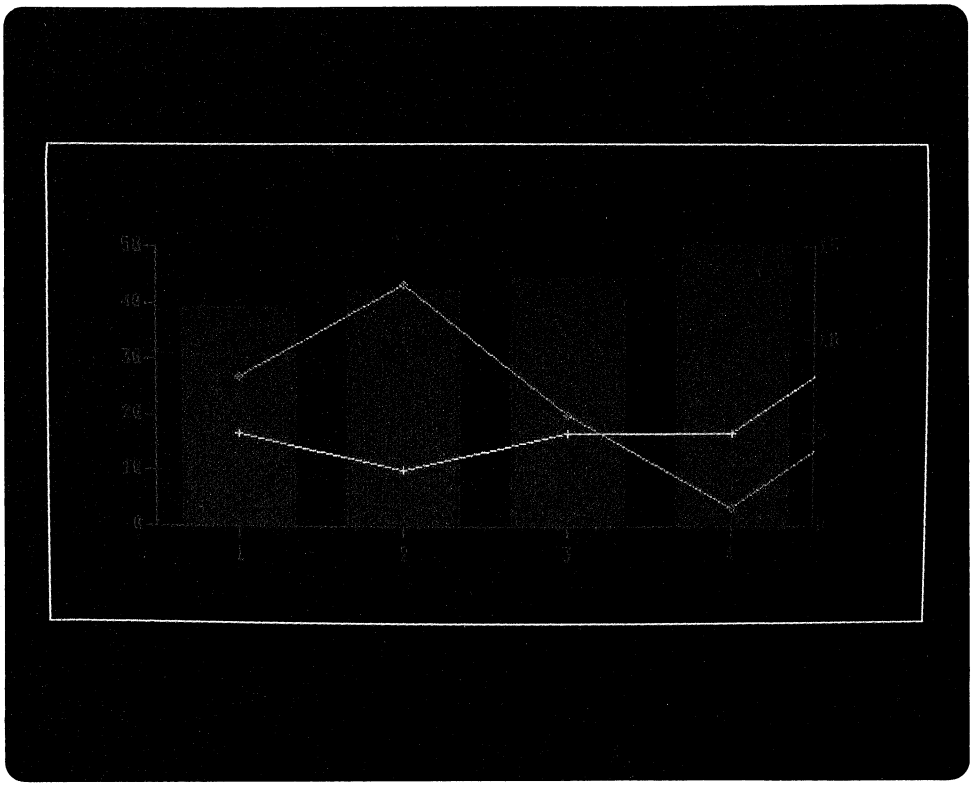
```
CALL GDDM ('CHSET','YDUP')  ! Specify duplicate y axis
```

When an axis is duplicated, it is identical to the original. For some charts however, it is desirable to have a different scale or different attributes for the second axis, especially in those cases where two different chart types are drawn in the same area. This type of axis is call a *secondary axis*.

To draw a secondary axis, use one of the following routines:

CHXSEL – x-axis selection
CHYSEL – y-axis selection

CHXSEL or CHYSEL specifies which is the current axis, either the primary or the secondary one. The current axis is the one affected when attributes are set and against which data is plotted when the chart-drawing routine is executed.



Secondary y-axis. The bar chart is plotted first, then the line chart is plotted against the secondary axis. Note the difference in the left y-axis scale used for the bar chart versus the right y-axis scale of the line chart.

```
! Plot bar chart
    .
CALL GDDM ('CHYSEL',2)    ! Select secondary y-axis
    .
! Plot line chart
```

Setting the Axis Attributes

CHAATT – Axis line attributes. CHAATT specifies the attributes for each axis line. Attributes that can be set are color, line type, and line width.

Like other attribute-setting routines, CHAATT uses an array of numbers that correspond to attributes. However, the array CHAATT uses can specify attributes for the primary x-axis, primary y-axis, secondary x-axis, and secondary y-axis at one time. The first group of three elements specifies attributes for the primary x-axis, the second group for the primary y-axis, and so forth. Therefore, to set attributes for a y-axis, (and not an x-axis), you must use an array of six elements, the first three of which are ignored.

By default, each axis line is a solid narrow line, shown in the default color.

Positioning the Axis

The point at which the x and y axis cross can also be altered. Each axis can intercept the other at the bottom, middle, or top. The default is at the bottom.

CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

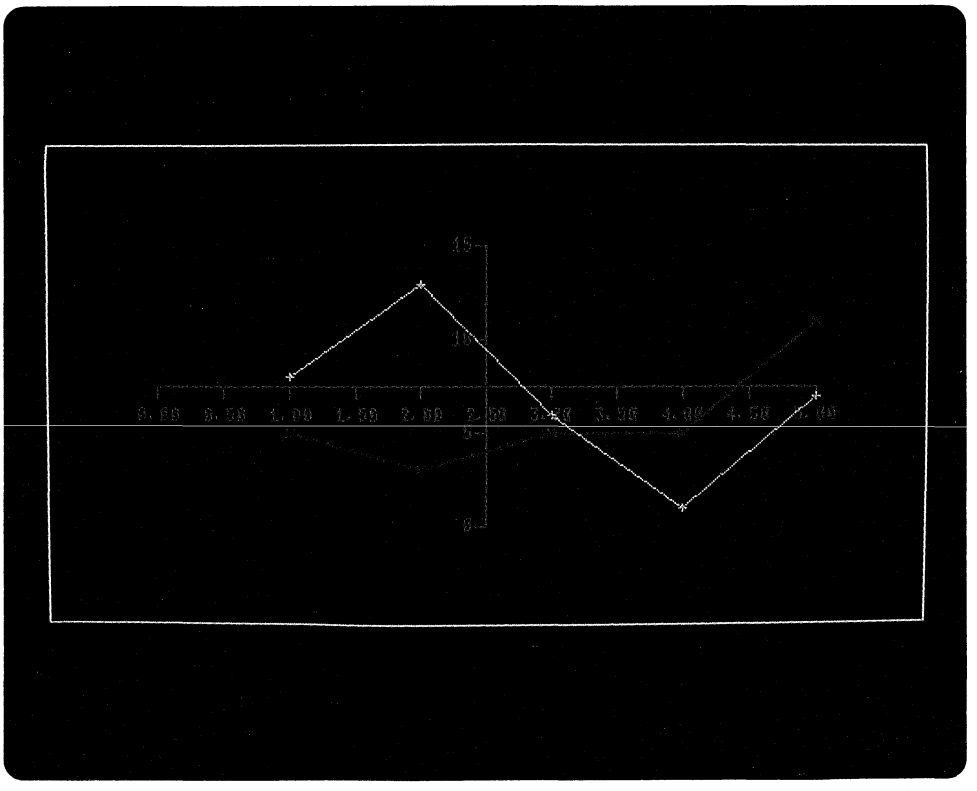
CHXSET or CHYSET (**LOWAXIS**|MIDDLE|HIGHAXIS|INTERCEPT) specifies the position of the axis in relation to the chart drawing area and/or in relation to the other axis. You can place the axis at the lower, middle, or upper end of the chart drawing area, measuring left to right or bottom to top. You can also specify whether you want the axis to appear as an intersecting axis, by specifying INTERCEPT and the appropriate CHXINT or CHYINT routine.

Axes are drawn at the left edge and the bottom edge of the chart drawing area by default.

CHXINT – x-axis intercept
CHYINT – y-axis intercept

CHXINT and CHYINT specify the point at which the other axis intercepts the named axis. For example, CHXINT gives the position where the y axis crosses the x axis and with it CHYSET(INTERCEPT) must be specified.

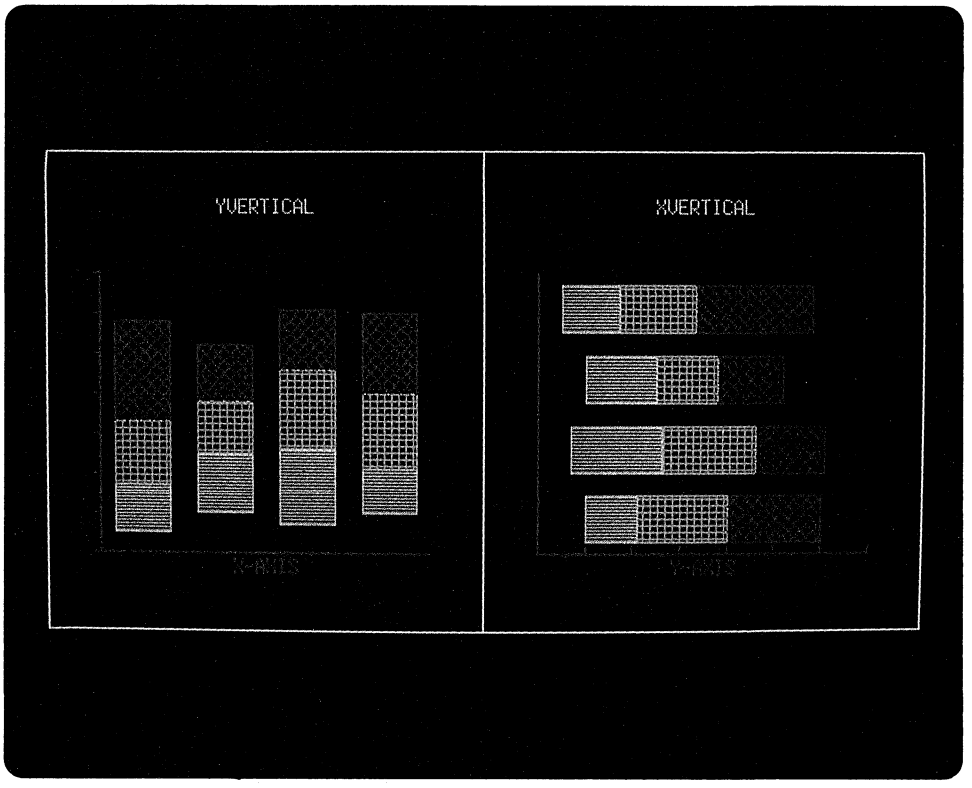
By default, the point of interception is zero; or, if some of the data used for an axis is negative, the point of interception is less than the lowest data value for that axis.



Quadrant. The quadrant results from axes that intercept each other at the middle of the range.

```
CALL GDDM ('CHXSET','INTERCEPT')  ! Allow intercept with y axis
CALL GDDM ('CHXINT',2.5)          ! y axis intercepts at 2.5
CALL GDDM ('CHYSET','INTERCEPT')  ! Allow intercept with x axis
CALL GDDM ('CHYINT',7.5)          ! x axis intercepts at 7.5
```

CHSET – Specify chart options. CHSET (**YVERTICAL**|XVERTICAL) specifies the orientation of the chart. You can change the normal orientation of the chart (x axis along the horizontal line) so that bar charts and histograms have horizontal bars, and multiple Venn diagrams and pie charts are drawn one above the other.



Horizontal orientation. The chart shows a floating bar chart with vertical orientation set by the YVERTICAL parameter, and one with horizontal orientation set by the XVERTICAL parameter.

```
CALL GDDM ('CHSET','YVERTICAL')  ! Orient chart vertically

CALL GDDM ('CHSET','XVERTICAL')  ! Orient chart horizontally
```

An entire chart can also be rotated in relation to the plotting surface on the plotter. For more information on GDDM device control routines, see Appendix A, "Devices Compatible with the AS/400 System."

Setting the Axis Range

Range is the upper and lower limit of measure used by each axis scale. By default, the range of the axis scale includes the upper limit of the data supplied; this is called *auto-ranging*. The default lower limit is zero or a negative number, depending on the data. An option exists to suppress zero as the lower limit, and instead use the lower limit of the supplied data. A line chart with an upper-limit value of 1984 on the x-scale and a supplied lower-limit value of 1980 would be difficult to interpret if the data were charted on a scale that ranged from 0 through 1984.

If you choose, you can override the system-generated range with your own range.

CHXRNG – x-axis explicit range
CHYRNG – y-axis explicit range

CHXRNG and CHYRNG set the range for the specified axis. If neither is specified, auto-ranging is used.

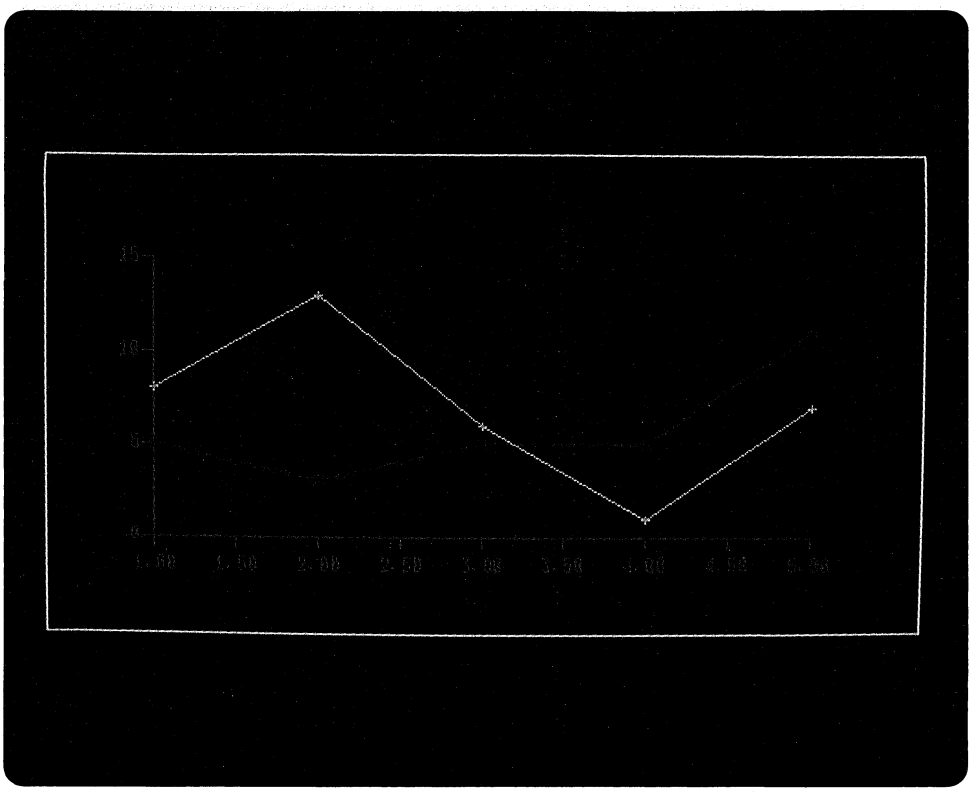
If you specify a range smaller than the range of your data, the areas of the components that exceed the range will be clipped.

CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

CHXSET or CHYSET (**FORCEZERO**|NOFORCEZERO) specifies whether auto-ranging includes zero. (Auto-ranging is the default if CHXRNG or CHYRNG have not been specified.) By default, zero is included as the lowest range value.

If all data values for the axis are positive, FORCEZERO makes zero the lower limit. If all data values for the axis are negative, FORCEZERO makes zero the upper limit. If some of the data values are positive and some are negative, FORCEZERO is ignored.

You can use CHXSET and CHYSET to write your own labels for the axes, or to write system-generated day or month labels. For these, FORCEZERO is ignored. For more information on writing your own labels, see page 4-36.



NOFORCEZERO on an x axis. When the lower limit of the supplied data is used for the lower limit of the range, the components begin at the y-axis reference line.

```
CALL GDDM ('CHXSET','NOFORCEZERO')
! Suppress zero as lower range limit for x axis
```

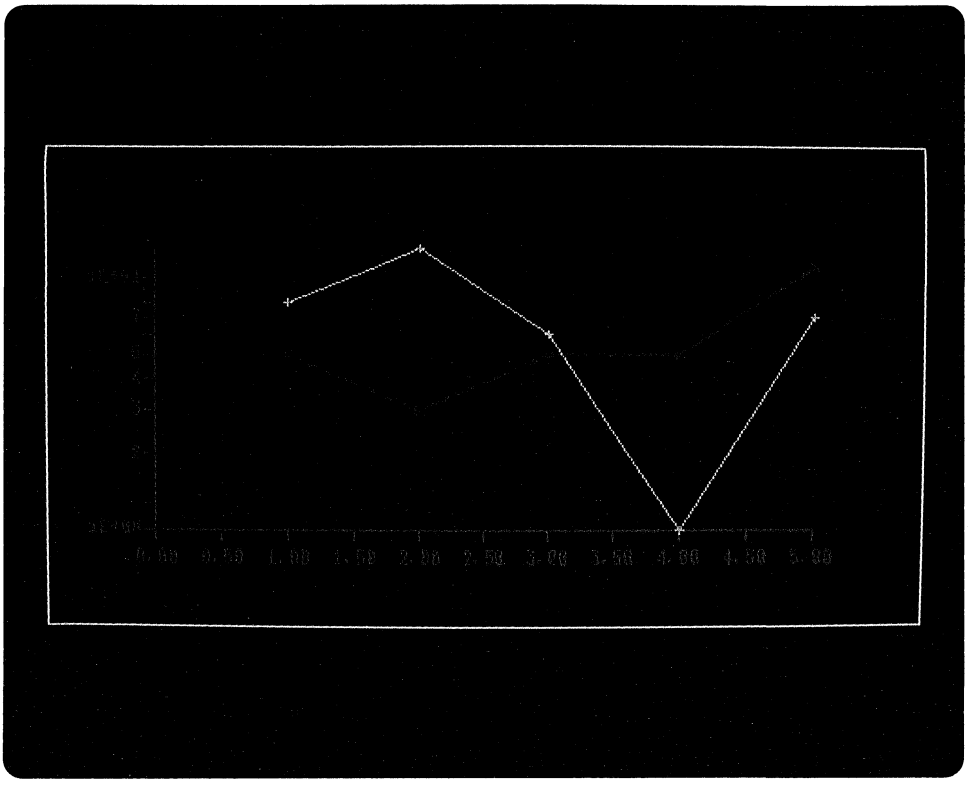
Setting the Axis Scale

The scale is the unit of measure applied to each axis, such as hours, meters, or dollars. Each scale can be linear (the default) or logarithmic. Linear scales are those where the progression of tick marks and labels is even and constant. Logarithmic scales use a progression where the tick marks are placed closer to each other as the scale values grow larger. Logarithmic scales are useful for charts with data that grows at an exponential rate.

CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

CHXSET or CHYSET (**LINEAR**|LOGARITHMIC) specifies the type of scale for the x or y axis. If ALPHANUMERIC labels are specified, LOGARITHMIC is ignored. (ALPHANUMERIC labels are described later on page 4-33.) Logarithmic scales are not valid for the x axis of bar charts. Data plotted against a logarithmic scale must be positive, nonzero and the labeling must be numeric.

The labels on a logarithmic axis scale are shown in floating-point notation, where 1E+01 = 10, 1E+02 = 100, 1E+03 = 1000, and so forth.



Logarithmic scale. The y axis shown is a logarithmic scale. Tick marks have been added with the CHYTIC routine (see page 4-28).

```
CALL GDDM ('CHXSET','LOGARITHMIC')    ! Use log scale
CALL GDDM ('CHYTIC',10.0,0.0)         ! Set tick mark interval
```

Drawing the Axis Tick Marks

Tick marks show intermediate values on the axes. Tick marks along each axis can make a chart much easier to interpret, by showing more points of reference.

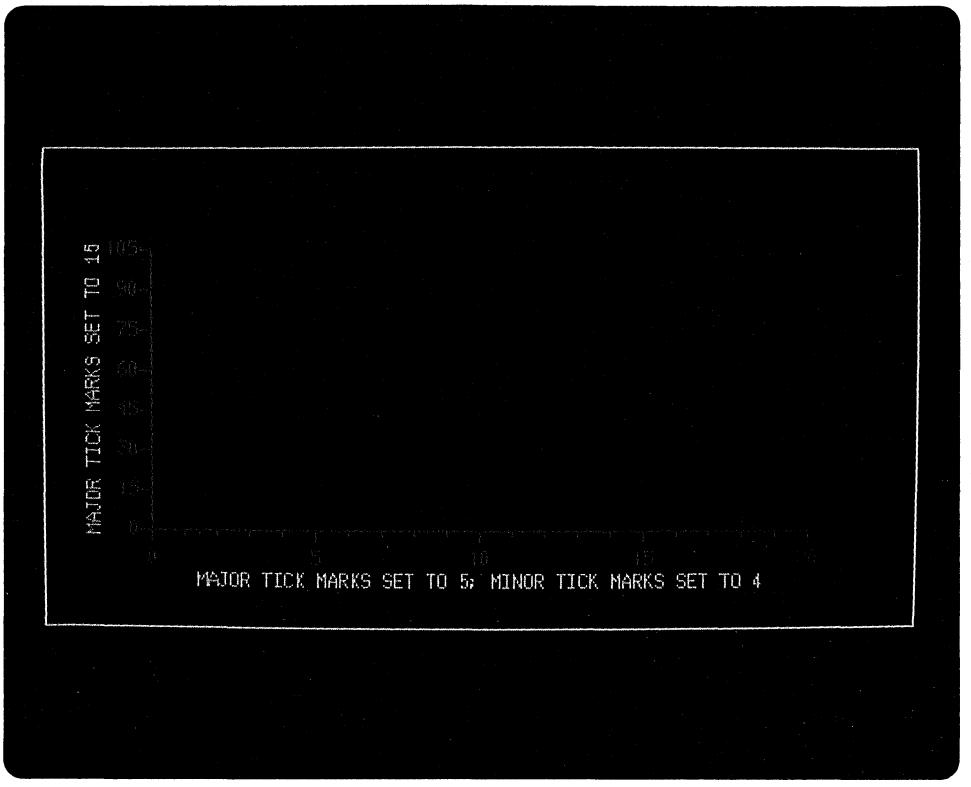
CHXTIC – x-axis tick mark interval
CHYTIC – y-axis tick mark interval

CHXTIC and CHYTIC set the level of incrementing between major tick marks, and specify the number of minor tick marks to place between each pair of major tick marks.

If neither is specified, the interval on a linear scale is 1, 2, or 5 multiplied by a power of 10, according to the auto-ranging values. The interval on a logarithmic scale is each power of 10: 1, 10, 100, 1000, and so forth.

With auto-ranging, a scale for a chart drawn on the 5292 Model 2 can be constructed with a larger interval between major tick marks than the same chart drawn on the plotter, because of space limitations. In other words, the major tick marks could be labeled 10, 20, and 30 on the display as opposed being labeled 10, 15, 20, 25, and 30 on the plotter.

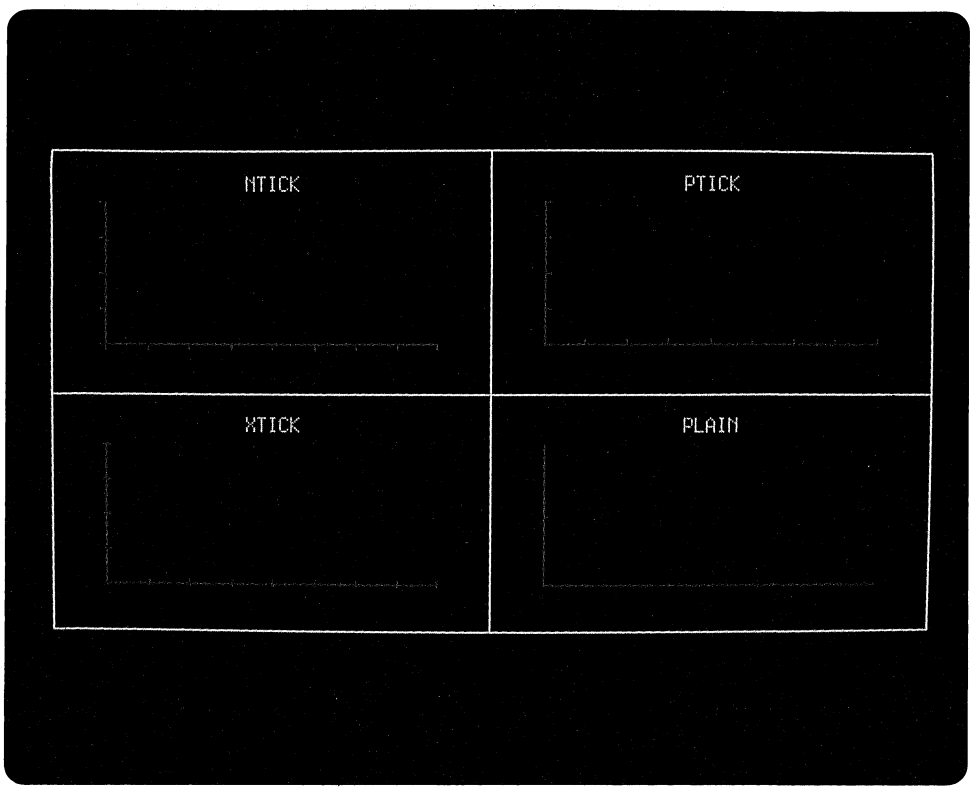
Major and minor tick marks. You can set the interval of the major tick marks relative to the range, and you can set the number of tick marks placed between major tick marks.




```
CALL GDDM ('CHXTIC',5.0,4.0)    ! x-axis tick mark interval
CALL GDDM ('CHYTIC',15.0,0.0)   ! y-axis tick mark interval
```

CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

CHXSET or CHYSET (**NTICK**|PTICK|XTICK|PLAIN) specifies the type of axis tick mark, or suppresses the tick marks. The type of tick mark specified for an axis is the same for major and minor tick marks, but minor tick marks are half the length of major ones.



Tick marks. Tick marks can be used to increase the usability of axis reference lines. These are the four options available.

Writing the Axis Text

You can add titles and labels to axes. Both are written with the default graphics symbol set or with the set loaded by the GDDM routine GSLSS. The size of the axis text characters is based on the size specified by CHCGRD, but the actual size is specified by the CHTATT and CHLATT routines.

Writing Axis Titles

Each axis of the chart can be assigned a title. By default, the x-axis title is centered below the x-axis reference line, and the y-axis title is centered to the left of the y-axis reference line.

Setting the Title Attributes

CHTATT – Axis title text attributes. CHTATT specifies the attributes for characters used in axis titles. Attributes that can be set are color, character mode, character symbol-set selection, and character size.

If you specify character-mode 2 instead of character-mode 3, the characters in the vertical axis title (usually the y-axis title) are oriented with a horizontal baseline and read from top to bottom. For either axis title, the character size multiplier for mode-2 characters serves only to set the spacing of the characters in the title. For mode-2 titles, the symbol-set specification is ignored.

If CHTATT is not specified, each axis title is shown in standard-size device default characters in character-mode 3, shown in the default color.

Writing the Axis Title

CHXTTL – x-axis title text specification
CHYTTL – y-axis title text specification

CHXTTL and CHYTTL specify the length of the text string to be used for the title, as well as the string itself.

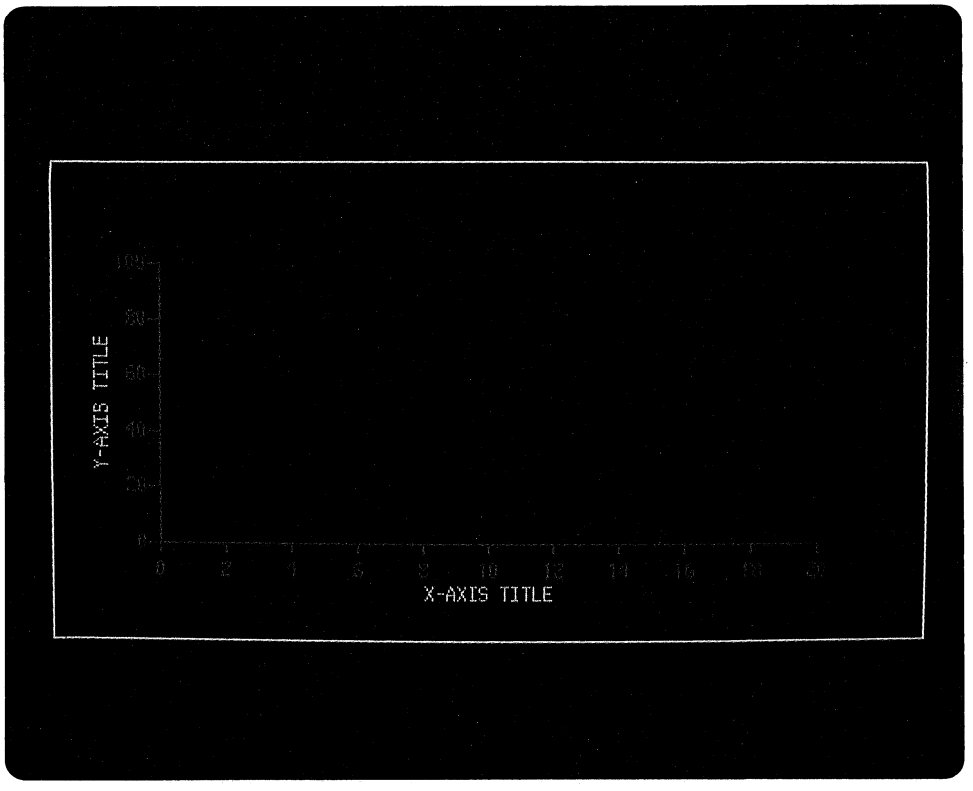
Positioning the Title

CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

CHXSET and CHYSET (**ATCENTER**|ATEND|ATABOVE) control the positioning of the axis titles.

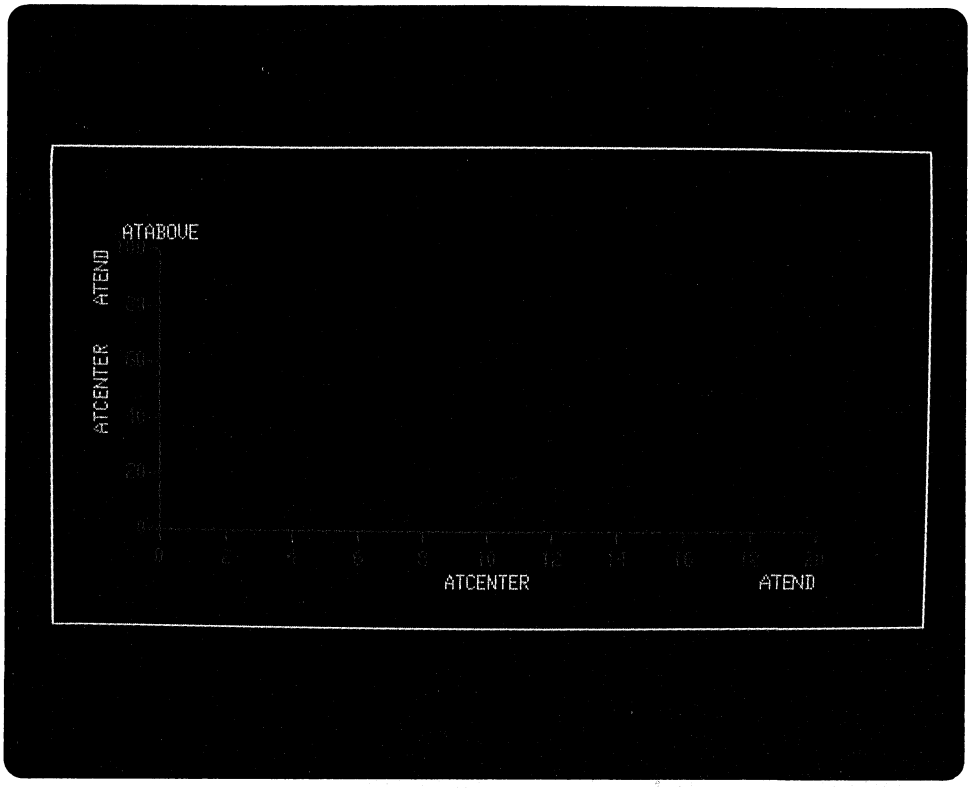
By default, the titles are centered on the axis.

Axis titles. When attributes for titles are *not* set by CHTATT, axis titles are mode-3 characters.




```
CALL GDDM ('CHXTTL',12,'X-AXIS TITLE')   ! Write axis title
CALL GDDM ('CHYTTL',12,'Y-AXIS TITLE')   ! Write axis title
```

Axis title placement options. Axis titles can be placed in the positions shown. The titles cannot be moved closer or further from the axis line. For that, you should suppress the title and use the chart notes instead.




```
CALL GDDM ('CHXSET','ATEND')   ! x-axis title at end of axis
```

Writing Axis Labels

Axis labels are the numbers or characters that correspond to major tick marks along the x- and y-axis scale. The default labels are the floating-point auto-ranging numbers determined from your data.

Labels correspond to major tick marks, but you can specify that the labels are positioned between the major tick marks. For an axis that shows a range of time values, or for a bar chart or a histogram, this type of label positioning can improve the chart by making the component values easier to understand.

You can specify your own labels (and position them at the same time), or you can suppress the labels altogether. A number of attributes can be specified for the labels. The size of the label characters is based on the size specified by the Presentation Graphics routine CHCGRD, but the specific size is specified by the CHLATT routine.

For character-mode-2 labels, the character size multiplier determines the spacing of the characters in each label, and the symbol set specification is ignored. Mode-3 labels are expanded or reduced by the multiplier.

For multiple-pie charts, you can specify labels to be used as titles for the pies.

Setting Label Attributes

CHLATT – Axis label text attributes. CHLATT specifies the attributes for characters used in axis labels and titles for individual pies on multiple pie charts. CHLATT affects both x- and y-axis labels. Attributes that can be set for labels are color, character mode, character symbol-set selection, character size, rotation, and height/width multiplier.

If CHLATT is not specified, each label uses unrotated, standard-size device default characters of the default color.

Use CHXLAT and CHYLAT to specify label text attributes for individual axes.

Setting Individual Axis Label Attributes

CHXLAT – x-axis label attributes
CHYLAT – y-axis label attributes

CHXLAT and CHYLAT specify the attributes for characters used in labels on the axes. Attributes that can be set are color, character mode, character symbol-set selection, character size, rotation, and height/width multiplier.

Use CHLATT if you want both sets of axis labels to look the same.

Positioning the Labels

CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

CHXSET or CHYSET (**LABADJACENT**|LABMIDDLE|NOLAB) specifies the position of the labels for the axes. The labels can be placed next to the tick marks (LABADJACENT), they can be placed between the tick marks (LABMIDDLE), or they can be suppressed (NOLAB).

Blanking the Label Area

CHSET – Specify chart options. CHSET (**NBLABEL**|BLABEL) specifies whether the areas where axis labels are positioned are blanked.

When the area is *blanked*, no other display feature can occupy the label text box (text boxes are explained on page 3-31). When the area is not blanked, the labels can *overpaint* the component. Label blanking does not apply to plotters.

By default, the areas are not blanked.

Specifying the Type of Label

CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

CHXSET or CHYSET (**NUMERIC**|DATE|ALPHANUMERIC) specifies the type of label for the axis. You can request that numeric labels are generated automatically, that date labels are used as specified by CHXMTH, CHYMTH, CHXDAY, or CHYDAY, or that alphanumeric labels are used (the labels you specify with CHXLAB or CHYLAB).

Numeric Labels Generated by the System

CHXSCL – x-axis scale factor
CHYSCL – y-axis scale factor

CHXSCL and CHYSCL specify a multiplier for numeric labels. (The default is 1.) You can use this multiplier to suppress leading or trailing zeros on labels. If you suppress zeros, you should word the title of the axis to reflect the value of the labels; for example, *SALES (IN THOUSANDS)*.

CHSET – Specify chart options. CHSET (**NPGFS**|PGFS) specifies the method of punctuating numbers greater than 1000. PGFS suppresses the punctuation except for the decimal point.

NPGFS uses OS/400 system value QDECFMT to specify the type of punctuation used by your system for numbers greater than 1000.

Month Labels Generated by the System

CHXMTH – x-axis month labels
CHYMTH – y-axis month labels

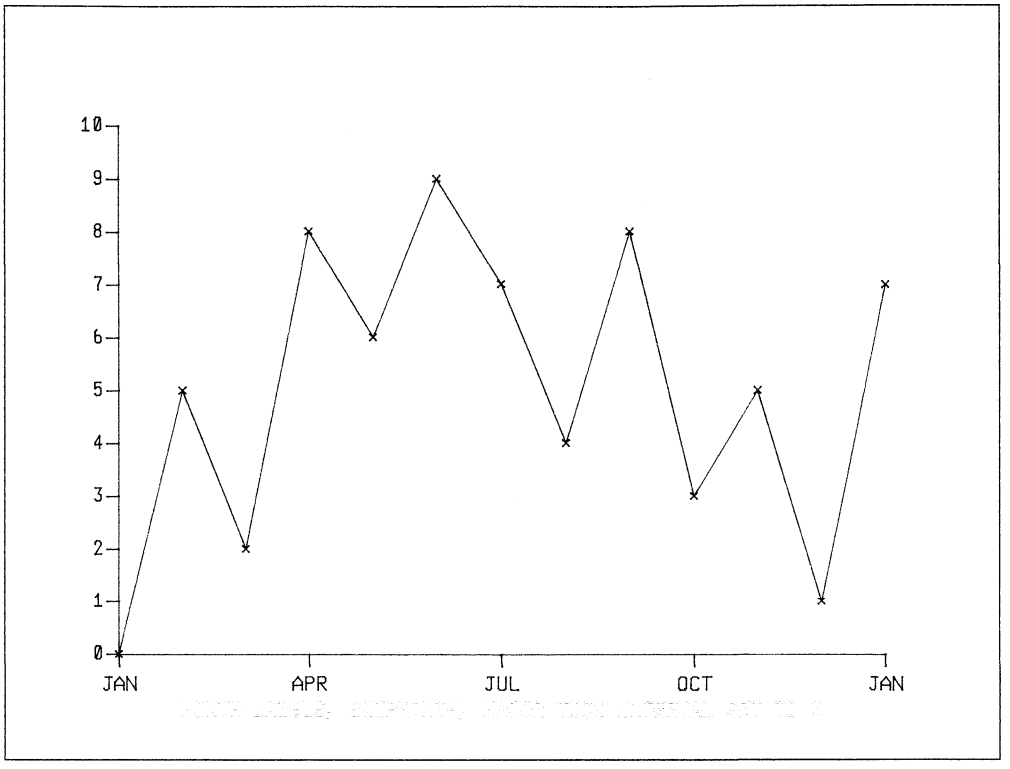
CHXMTH and CHYMTH specify the successive months to be used as labels, in terms of 1,2,3,... (for JAN,FEB,MAR, and so forth). If you specify CHXMTH,2, FEB is the first label.

You can use CHSET (**ABREV**|FULL|LETTER) to select the appearance of the month labels.

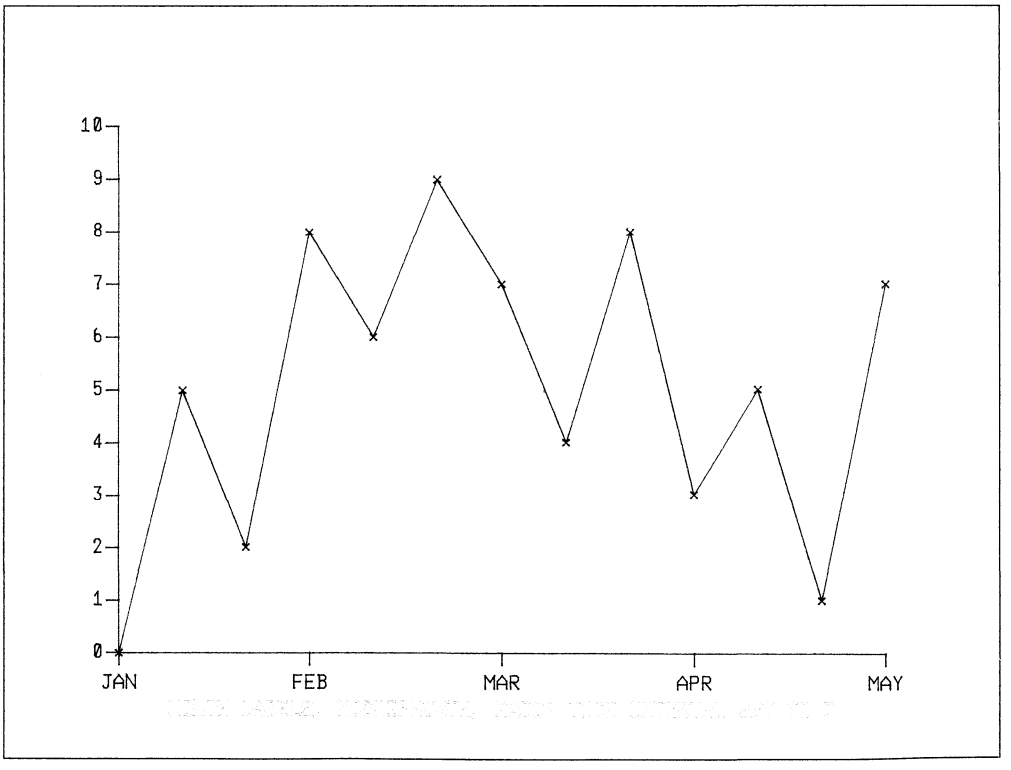
CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

CHXSET or CHYSET (SKIPMONTH|**NOSKIPMONTH**) specifies how month labels will be displayed at the major tick marks.

NOSKIPMONTH specifies that consecutive months are used for each successive tick mark. This is the default.



SKIPMONTH specifies that month labels are not used consecutively. Instead, they are selected to match the major tick interval specified.



When you specify month labels, if the range has more than 12 major tick marks and NOSKIPMONTH is specified, the month labels are reused as needed.

Day Labels Generated by the System

CHXDAY – x-axis day labels
CHYDAY – y-axis day labels

CHXDAY and CHYDAY specify the successive days to be used as labels, in terms of 1,2,3,... (for MON,TUE,WED and so forth). If you specify CHXDAY,2, TUE is the first label. Consecutive days follow for each major tick mark.

You can use CHSET (**ABREV**|FULL|LETTER) to select the appearance of the day labels.

When you specify day labels, specify CHXTIC or CHYTIC,1 for the labels to correspond one for one to the major tick marks in the range. If the range has more than seven major tick marks, the day labels are reused as needed.

CHSET – Specify chart options. CHSET (**ABREV**|FULL|LETTER) specifies the form of axis labels supplied by the CHXMTH, CHYMTH, CHXDAY, and CHYDAY routines. The names can appear in 3-character abbreviations, the full names, or the first letter of the name.

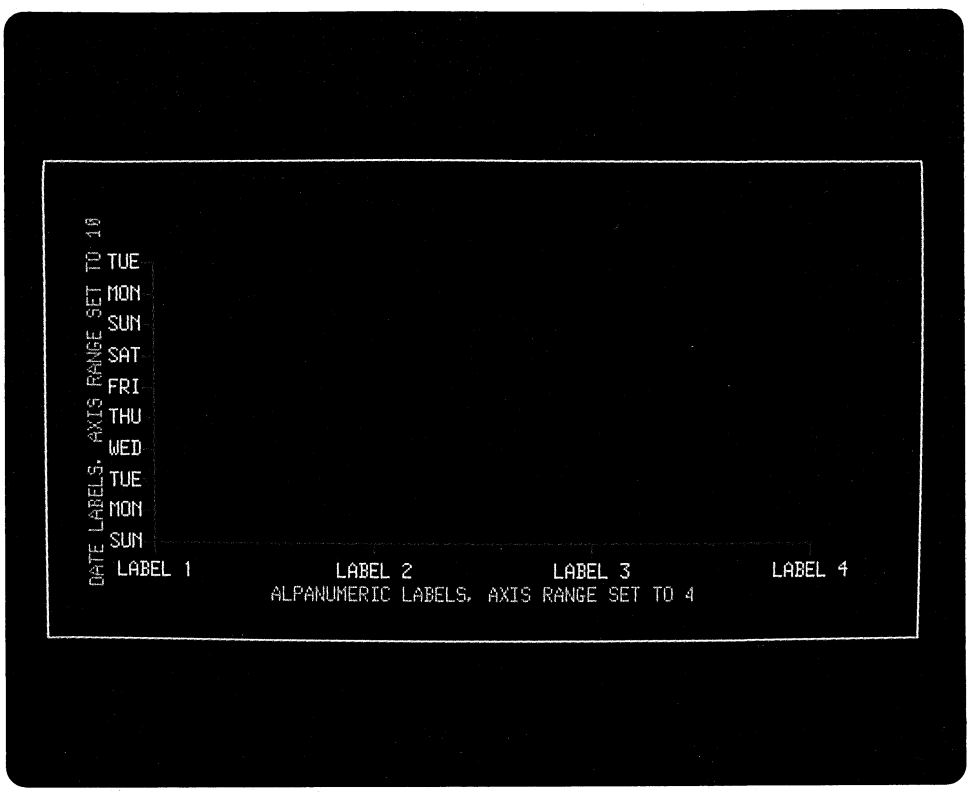
Your Own Labels

CHXLAB – x-axis label text specification
CHYLAB – y-axis label text specification

CHXLAB and CHYLAB specify the number of labels to be used, the length of the text string to be used for the label, and the label text itself.

When you specify your own labels, you should make sure that the range (specified by auto-ranging or by CHXRNG and CHYRNG) for the axis matches the number of labels you have supplied. For example, if you provide seven labels for a range that has five major tick marks, the last two of your labels are ignored. If you provide eight labels for a range that has 10 major tick marks, the first two of your labels are reused for the ninth and tenth tick marks. Use CHXRNG and CHYRNG to set the range, and use CHXTIC and CHYTIC to set the interval for the tick marks.

Axis labels. Labels are associated with the major tick marks of the range.




```
CALL GDDM ('CHXLAB',4,7,'LABEL 1LABEL 2LABEL 3LABEL 4')
! Use 4 7-character labels as x-axis labels
CALL GDDM ('CHXRNG',1.0,4.0)
! Set lower x-axis range limit = 1, upper limit = 4
CALL GDDM ('CHXTIC',1.0,0.0)
! Use 1 major tick mark for every range value
          .
          .
CALL GDDM ('CHYDAY',7)
! Use day labels for the y axis, starting with SUN
CALL GDDM ('CHYRNG',1.0,10.0)
! Set lower y-axis range limit = 1, upper limit = 10
CALL GDDM ('CHYTIC',1.0,0.0)
! Use 1 major tick mark for every range value
```

Drawing Other Reference Lines

Drawing Grid Lines

Grid lines are extensions of tick marks that are drawn across the component-plotting area. Grid lines can be drawn from the x axis parallel to the y axis, or can be drawn from the y axis parallel to the x axis.

Grid lines can be useful for charts where the components cannot be compared easily to the axis scale, such as scatter plots where no interconnecting lines exist to help the user judge the relative x- and y-values of the markers. In some charts, grid lines can reduce the usefulness of the chart by making it look complicated and busy. For such charts, try using a duplicate axis (CHSET (XDUP or YDUP)) to increase the usability of the chart.

If you must use a grid with a secondary axis (CHXSEL or CHYSEL) that uses a scale that differs from the primary axis, you should use CHAATT and CHGATT to change the color or line type of the axis and grid line to differentiate those features from the primary axis.

Setting Grid Line Attributes

CHGATT – Grid line attributes. CHGATT specifies the attributes for grid lines. Attributes that can be set are color, line type, and line width.

Like other attribute-setting routines, CHGATT uses an array of numbers that correspond to attributes. However, the array CHGATT uses can specify attributes for grids for the primary x axis, primary y axis, secondary x axis, and secondary y axis at one time. The first group of three elements specifies attributes for the primary x-axis grid, the second group for the primary y-axis grid, and so forth. Therefore, to set attributes for a y-axis grid (and not an x-axis grid), you must use an array of six elements, the first three of which are ignored.

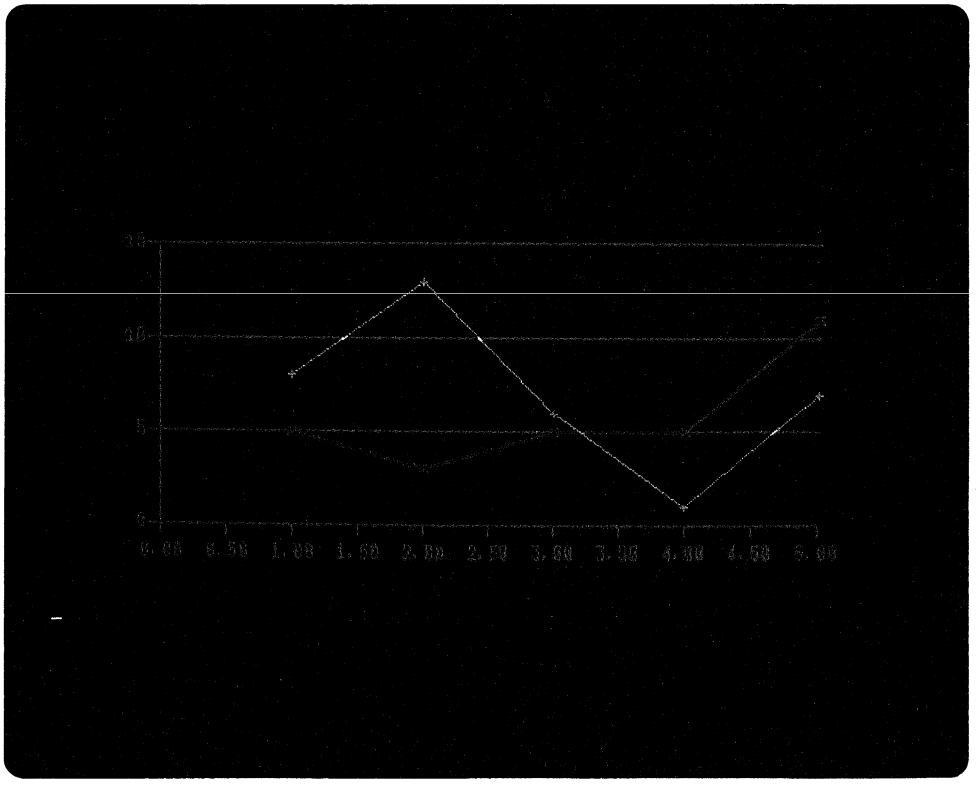
If CHGATT is not specified, each grid line is a solid narrow line, shown in the default color.

Drawing Grid Lines

CHXSET – x-axis characteristics
CHYSET – y-axis characteristics

CHXSET or CHYSET (**NOGRID**|GRID) specifies whether grid lines perpendicular to the axis are drawn.

By default, the grid lines are not drawn.



Grid lines. The grid lines for this chart could be supplemented with minor tick marks for the y axis (CHYTIC).

```
CALL GDDM ('CHYSET','GRID')   ! Use y-axis grid
```

Drawing Translated Axis Lines and Datum Lines

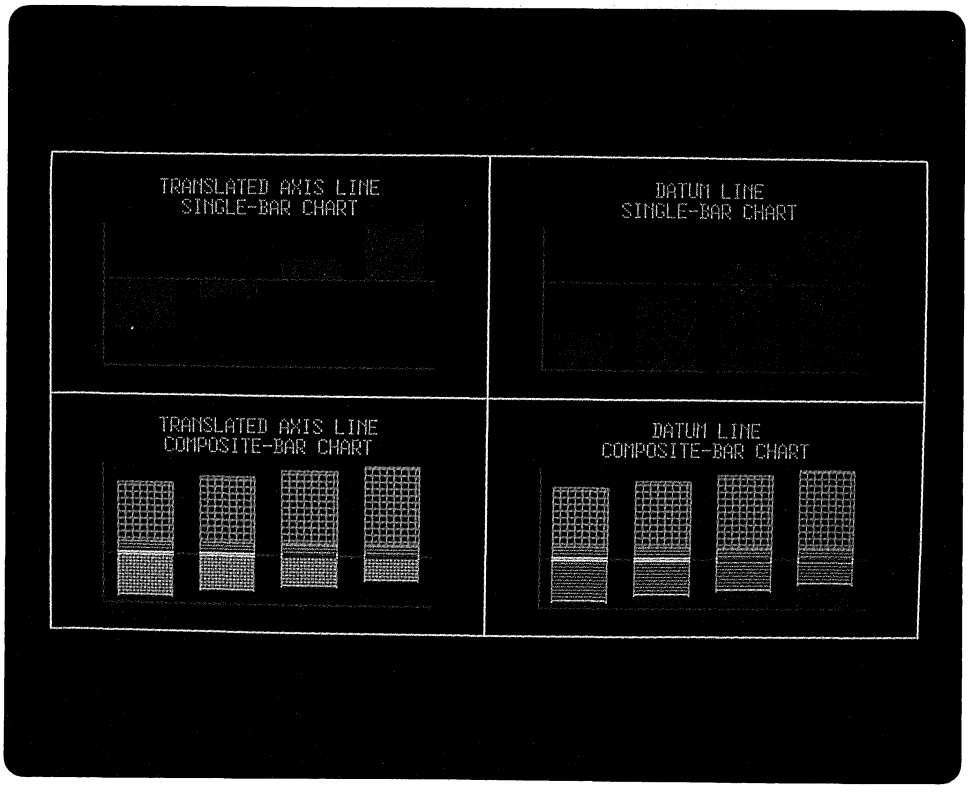
Drawing Translated Axis Lines

Translated axis lines are drawn *before* the components that represent data groups are drawn. Then, when the components for the chart are drawn, (in state-1) they are *based* on the translated axis line. In other words, the first component of a composite-bar chart, surface chart, or histogram has values originating from the translated axis line rather than the x axis. This lets you show quantities that extend both up and down from the translated axis line.

Drawing Datum Lines

Datum lines are similar to grid lines, except that you can only have one datum line from a specified axis value. Datum lines are drawn *after* the components are drawn (in state-2). They function as reference lines (much like a single line from a grid).

The following illustration shows the difference between a translated axis line and a datum line.



Translated axis line versus datum line. Because CHYDTM is called in state-1, it results in a translated axis line. In state-2, the datum line results. Both charts use the same data.

```
! State-1
CALL GDDM ('CHYDTM',25.0)
! Draw y-axis translated axis line from y=25 scale value
! ----(Chart-drawing routine)----
! State-2
        .
        .
! State-1
! ----(Chart-drawing routine)----
! State-2
CALL GDDM ('CHYDTM',25.0)
! Draw y-axis datum line from y=25 scale value
```

Setting Translated Axis Line or Datum Line Attributes

CHDATT – Datum line attributes. CHDATT specifies the attributes for translated axis and datum lines. Attributes that can be set are color, line type, and line width.

If CHDATT is not specified, each line is a solid narrow line, shown in the default color.

Drawing Translated Axis Line or Datum Line

CHXDTM – x-datum line
CHYDTM – y-datum line

CHXDTM and CHYDTM specify a point from which a translated axis line or datum line is drawn. For example, CHYDTM(10) draws a line that corresponds to the value 10. The line is drawn parallel to the x axis.

If CHXDTM or CHYDTM is specified *before* the chart is drawn (state-1), the resulting line is a translated axis line. If CHXDTM or CHYDTM is specified *after* the chart is drawn (state-2), the resulting line is a datum line.

Drawing Chart Legends

Chart legends provide a means of identifying the components shown on the chart. The legend matches the *legend key label* of the component with an identifying characteristic of the component, such as color, marker, or pattern. This identifying characteristic is called the *key*; it is the name or value associated with the component or the component key.

Legends can be placed anywhere on the chart, and a number of attributes can be assigned to specify the appearance of the legend.

Drawing or Suppressing the Legend

CHSET – Specify chart options. CHSET (**LEGEND**|NOLEGEND) specifies that a legend is to be constructed (LEGEND). CHSET (NOLEGEND) suppresses the legend.

For a program in which multiple calls to a chart-drawing routine are used to draw separate chart components, CHSET (NOLEGEND) suppresses the legend for each of the chart-drawing routines. Then, to draw a legend that includes legend keys and

labels for each of the chart components, specify CHSET (LEGEND) before the last call to the chart-drawing routine.

CHSET – Specify chart options. For pie charts, CHSET (**PIEKEY**|SPIDER) specifies whether a legend should be drawn (PIEKEY) or, if labels are to surround the pie, whether each is connected to the associated pie slice with a *spider tag* (SPIDER).

Positioning the Legend

CHKEYP – Legend position. CHKEYP specifies the orientation of the legend (vertical or horizontal), and its position in a margin.

The default is a vertical legend centered in the right margin.

CHKOFF – Legend offsets. CHKOFF specifies the final position of a vertical or horizontal legend. The legend is based on the position specified by the CHKEYP routine, and moved a positive or negative number of character units to its final position.

By default, no offsets are applied.

CHKMAX – Maximum legend width/height. CHKMAX specifies the maximum height of a vertical legend, or the maximum width of a horizontal legend.

By default, the maximum dimensions for either legend orientation is limited by the dimension of the chart area.

CHSET – Specify chart options. CHSET (**KNORMAL**|KREVERSED) specifies the order of items listed in the legend. The normal order (KNORMAL) is for the first entry to be listed at the bottom of the legend and the last entry at the top. KREVERSED reverses that order. If the legend is horizontal, KNORMAL lists the first entry to the left.

Blanking the Legend Area

CHSET – Specify chart options. CHSET (**NBKEY**|BKEY) specifies whether an area is blanked or not blanked before the legend is drawn in that area. (On a plotter, this option has no effect.)

By default, the legend area is not blanked.

Legend area blanking. The legend area is blanked so that the legend is not obscured by the patterns of the components.

```
CALL GDDM ('CHSET','BKEY')  ! Blank the legend area
```

Enclosing the Legend in a Box

CHSET – Specify chart options. CHSET (**NKBOX**|KBOX) specifies whether the legend is enclosed in a box.

By default, the legend is not enclosed in a box.

Writing the Legend Key Labels

CHKATT – Legend text attributes. CHKATT specifies the attributes to be used for legend key labels, pie chart spider key labels, and labels on a Venn diagram written by CHKEY. The attributes that can be specified are color, character mode, symbol set, and character size.

If CHKATT is not specified, labels are written in the default character set with the standard size characters for the device, in the default color.

CHKEY – Legend key labels. CHKEY identifies the labels to be associated with the legend keys. For a pie chart, the labels specified here appear adjacent to the pie slices when CHSET (SPIDER) is specified; otherwise, the labels appear in a legend (the default).

Note: Legends cannot be constructed for Venn diagrams; the legend text (CHKEY) appears as a label adjacent to each population.

Writing Chart Notes

Chart notes are character strings that are positioned anywhere on a chart to explain or identify an aspect of the chart. You can write as many notes on your chart as you want. You can write a note, then select new attributes and write a different note elsewhere on the chart. Notes are written with the graphics symbol set loaded by the GDDM routine GSLSS. The size of the label characters is based on the size specified by the Presentation Graphics routine CHCGRD, but the specific size is specified by the CHNATT routine.

When you use notes, be careful that they add to, not detract from, the simplicity and usefulness of the chart.

Chart notes can be written only in state-2. (State-2 is any point *after* the chart drawing routine is executed.)

Setting Attributes for Notes

CHNATT – Specify text attributes for notes. CHNATT specifies the attributes to be used by notes. The attributes that can be specified are color, character mode, symbol set, character size, rotation, and height/width multiplier.

If CHNATT is not specified, notes are shown in the default color, and are written unrotated in the default mode-3 character set with the standard size characters for the device.

Blanking the Note Area

CHSET – Specify chart options. CHSET (**NBNOTE**|BNOTE) specifies whether an area is blanked or not blanked before the note is written in that area. (On a plotter, this option has no effect.)

Enclosing the Note in a Box

CHSET – Specify chart options. CHSET (**NONBOX**|NBOX) specifies whether the note is enclosed in a box.

Writing the Note

CHNOFF – Specify offsets for CHNOTE. CHNOFF specifies the position of a note.

CHNOTE – Write a character string at a designated location. CHNOTE writes a note at a position specified by CHNOFF.

You can use a semicolon (;) as a line-break character to generate multiple line notes. To include a semicolon as part of a note, use two semicolons (;;).

The CHNOTE routine has three parameters:

The base position of the note
The number of characters in the note
The note text itself

The base position parameter value has an effect on the way the offsets specified by CHNOFF are interpreted. The base position value is a 2-character value, whose first character can be:

C The CHNOFF offsets are interpreted as device rows and columns.

H The horizontal offset is in chart axis units, while the vertical offset is in device columns.

V The vertical offset is in device columns while the horizontal offset is in device rows (from the bottom).

Z Both offsets are in chart axis units.

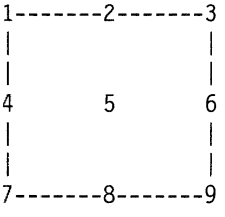
When the offset is in device rows and columns, the origin (0,0 offset) is considered to be the lower left corner of the chart area.

When the offset is in chart axis units, the origin corresponds to the zero value for the axis (the y axis for vertical offsets and the x axis for horizontal offsets, except for XVERTICAL orientation). An offset specified in axis units ensures that the note will appear in the same position on the chart when the chart is drawn on the plotter or when a different chart area is used.

The plotter uses different values for the character grid, so a note offset in device rows and columns *can* appear in a different location (relative to the chart axes) on the plotter from that on the display.

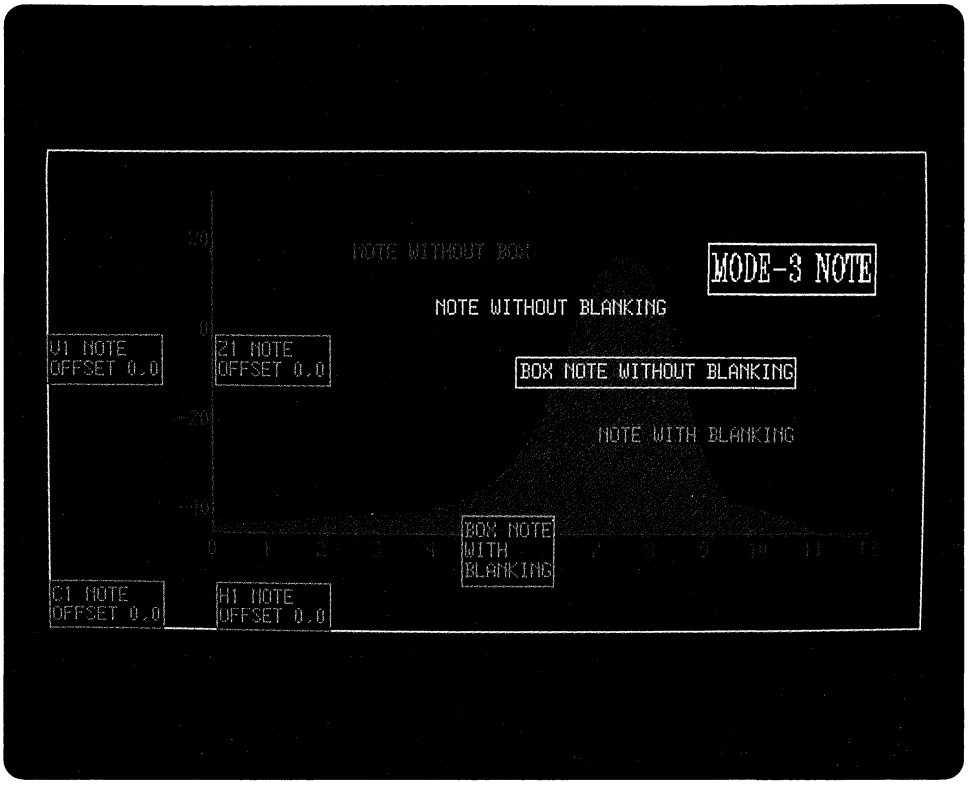
A note positioned in chart axis units is always drawn relative to the axes; its position will not change when the character grid units are changed (CHCGRD), when the chart area is changed (CHAREA), or when the chart margins change the chart area (CHHMAR or CHVMAR). The disadvantage in using chart axis offsets is that you cannot surround the chart with notes; negative offsets can be specified, but only to the extent of the chart axis units.

The second character of the base position parameter of CHNOTE specifies which part of the note box is placed at the offset position. The value for this part of the base position parameter is a number 1 through 9, where the number corresponds to these positions of the note box:



If the base position is C7, the note is positioned by row/column offset values with the lower left corner of the note box at the position specified by the offset.

Chart notes. The chart notes in red use an offset of 0,0 (the left margin is increased to 15 to preserve the y-axis scale for this illustration). The C1 note offset in device units appears despite having a base position that should place it outside the chart area.




```
! For the red chart notes:
CALL GDDM ('CHYRNG',-45.0,30.0)
CALL GDDM ('CHXRNG',0.0,12.0)
INTEGER NATT                  ! Declare integer
DIM NATT(2) : MAT READ NATT   ! Read note array
DATA 2,2
! 2 = red, 2 = character mode
CALL GDDM ('CHNATT',2,NATT())  ! Set note attributes
CALL GDDM ('CHNOFF',0.0,0.0)   ! Set note position
CALL GDDM ('CHNOTE','C1',18,'C1 NOTE;OFFSET 0,0')
CALL GDDM ('CHNOTE','H1',18,'H1 NOTE;OFFSET 0,0')
CALL GDDM ('CHNOTE','V1',18,'V1 NOTE;OFFSET 0,0')
CALL GDDM ('CHNOTE','Z1',18,'Z1 NOTE;OFFSET 0,0')
```

Designing the Chart Layout: Summary

The part you have just finished reading showed you how to set up the layout for a chart. In setting up this layout you can specify the following features:

Chart size, which also depends on the size of the chart character grid and chart margins

Chart frame or background

Chart heading

Chart reference lines (except for pie charts and Venn diagrams), including:

Axes, with tick marks, labels that correspond to tick marks, and the axis title
Grid lines
Translated axis or datum lines

Chart legend

Chart notes

You can set the chart size by specifying a chart area, by changing the size of the character grid on which the chart area is based, and by changing the size of the chart margins. Reducing the size of the margins increases the size of the chart, and the converse.

You can add a frame that encloses the chart and you can specify a background color for the chart.

To increase the usability of the chart, you can add a heading and chart reference lines. The chart reference lines can be added or changed to clarify the chart data, or suppressed to reduce clutter and to give the chart a simple appearance.

A legend and chart notes can be added to any type of chart, also to increase chart usability.

Except for chart size, you can specify attributes for any of the chart features. These attributes continue to be used until changed or until the Presentation Graphics environment is terminated or reinitialized (explained later).

The next part explains the routines you use to draw any type of chart and the routines that are specific to each chart type.

Drawing the Chart

After your chart-drawing program was initialized, you defined the appearance of your chart. The appearance of the chart was determined by routines you specified to change the defaults for:

The layout of the chart, including:

- Chart area
- Chart margins
- Chart frame
- Character size

The attributes of the chart, including:

- Chart headings
- Reference lines (including axes and axis text)
- Chart legends
- Chart notes

Now that you have defined the layout and attributes of the chart, you can call the routine that draws the chart, using attributes and the available data groups.

The data groups can be coded into the program, so that the chart looks the same each time you run the program, or the data groups can come from database files, so the chart reflects up-to-the-minute data.

Each data group must contain the same number of y values as there are x values. If one or more values is missing from your data, you should supply a dummy value of 1E35 (10^{35}) for all high-level languages except RPG/400, which requires a value of 1E20 (10^{20}). These dummy values are called *missing values*.

Note: Missing values are not supported for Venn diagrams.

Using Component Attributes

If your chart represents more than one data group, you need to differentiate the groups. For example, a line chart that shows three lines, all of the same line type and color would be very difficult to understand. Therefore, various *component attributes* can be assigned to distinguish one data group from another.

When a multiple component chart is drawn, some types of component attribute are selected from a table. These attributes are:

Color
Line type
Shading pattern
Marker type

Tables contain entries that determine the order the attributes are assigned. For example, a color attribute table could contain entries in this order: red, yellow, blue, and the default color. If the chart had three components, the first would be red, the second yellow, and the third blue. If the chart had five components, the fourth would be the default color and the fifth would be red (the table entries are reused in order). Note that the GDDM routines GSCOL, GSLW, GSPAT, and GSMARK are *not* used for Presentation Graphics.

The component attributes are discussed with each chart type.

Some of the following charts are pictures as produced on the display and some are plots as produced on the IBM 6180 Plotter. For those, the accompanying programs include the device routines necessary to send the picture to the plotter. For more information on device routines, see Appendix A, "Devices Compatible with the AS/400 System."

Drawing Line Charts

Setting the Color Selection Order

CHCOL – Set component color. CHCOL sets the color of the lines, by defining a table that holds the number of colors and the order of their selection.

If CHCOL is not specified, the sequence of colors in the default color table is used. The default color table is the GDDM color table which is either the default color table for the current page or a color table modified for use in the current page.

Setting the Line Type Selection Order

CHLT – Set component line type. CHLT sets the line type of the components by defining a table that holds the number of line types and the order of their selection.

If CHLT is not used, all components are drawn with the line type defaulted to or specified for entry 0 of the line type table. The default line type table shown for the discussion of the GSLT routine in Chapter 3, "Using GDDM" shows the order of selection. The components are differentiated by their colors and marker types.

Setting the Line Width

CHLW – Set component line width. CHLW sets the line width of the components by defining a table that holds the multiplication factor to be applied to the default line width.

If CHLW is not used, all components are drawn with the standard line width for the device.

Setting the Marker Type Selection Order

CHMARK – Set component marker. CHMARK sets the type of marker used by each of the components by defining a table that holds the number of markers and the order of their selection.

If CHMARK is not specified, the sequence of markers in the default marker table is used. The default marker table shown for the discussion of the GSMARK routine in Chapter 3, "Using GDDM" shows the order of selection.

Suppressing the Markers

CHSET – Specify chart options. CHSET (NOMARKERS) suppresses the markers.

Setting the Line Curve

CHSET – Specify chart options. CHSET(CURVE) draws the components with line curving; the degree of line curving is specified by the CHFINE routine. If CHSET(CURVE) is specified, the default value of CHFINE is 10.

A curved line is more attractive than one with straight lines connecting data points, but the curved line can be misleading on a chart where a high degree of accuracy is needed. Presentation Graphics passes the curved line *through* the data points, but does not interpolate the true values that lie between data points; therefore, values on the curved line between data points can be inaccurate.

CHFINE – Set line curving smoothness. CHFINE sets the degree of curve smoothness of the components. The higher the number specified for CHFINE, the smoother the curve drawn by CHSET(CURVE).

By default, the degree of smoothness is 10. A number higher than 10 might not add any noticeable degree of smoothness to the curve but it could greatly increase the time needed by the system to process and draw the chart.

Writing Data Values

CHSET – Specify chart options. CHSET(NOVALUES|VALUES) controls how values are displayed. CHSET(VALUES) writes the data value represented by each data point adjacent to the data point. CHSET(NOVALUES) indicates that no data values are displayed for the data points.

CHSET(BVALUES) blanks the areas where data values are written. When the area is blanked, no other display feature can occupy the data value text box. When the area is not blanked (NBVALUES), the values can overpaint the component.

CHVCHR – Number of data value characters. CHVCHR sets the number of characters to be used for showing data values on a line chart. The value text attributes are controlled by CHVATT.

Up to 15 characters can be used, but some of the 15 characters from one data value would probably overwrite those for the next data point.

By default, the true value represented by the data point is used, up to 9 character positions.

CHSET – Specify chart options. CHSET(NPGFS|PGFS) specifies the method of punctuating numbers greater than 1000 displayed on the chart. PGFS suppresses the punctuation except for the decimal point. NPGFS uses OS/400 system value QDECFMT to specify the type of punctuation used for numbers greater than 1000 by your system.

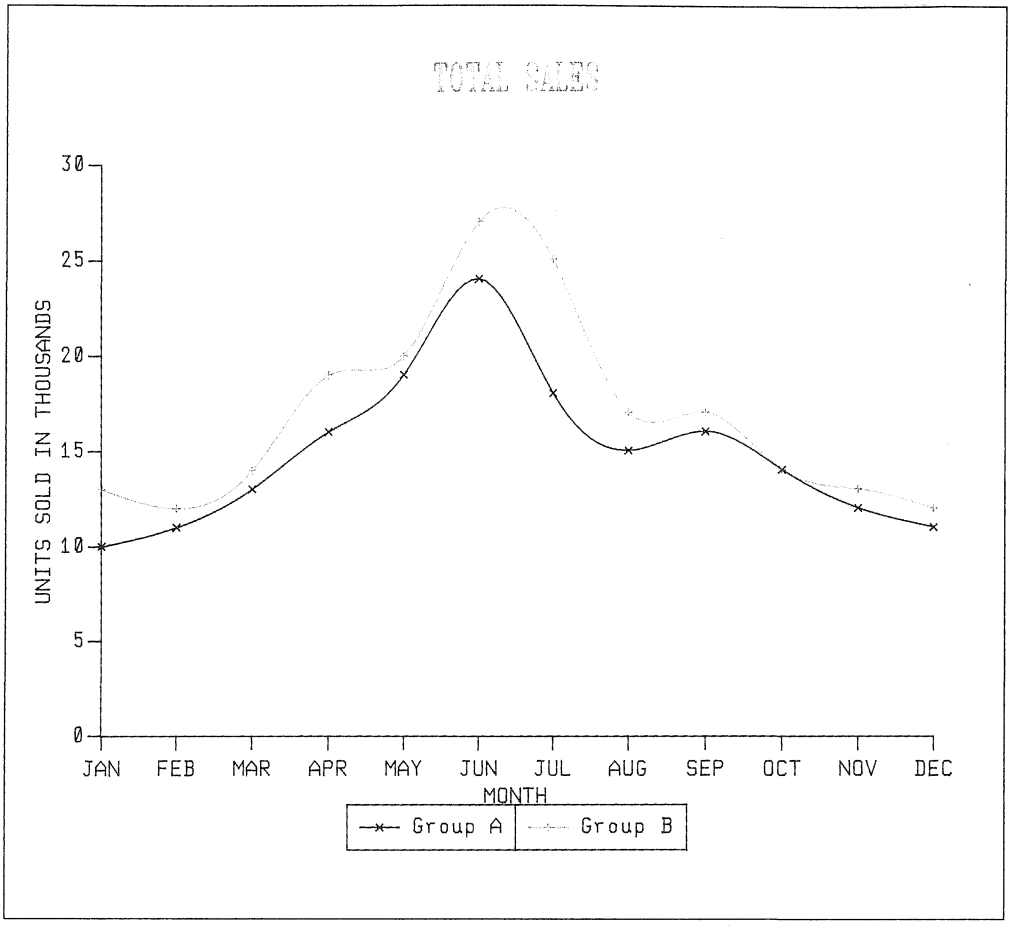
CHVATT – Value text attributes. CHVATT specifies the attributes for value text. Value text is used to show individual values for data points on a line chart. Attributes that can be set are color, and character font, size, and rotation.

If CHVATT is not specified, each data value is shown in standard-size default characters, in the default color, in the default 0 rotation.

Drawing the Chart

CHPLOT – Draw a line graph or a scatter plot. CHPLOT draws the line chart. Parameters in the CHPLOT routine specify the number of components to be drawn, the number of data points in each component, and the arrays used for the data groups the components are drawn to represent.

This is a line chart produced on a plotter:




```
00010 ! ***************** TOTAL SALES ****************************
00020 ! ****************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ************** Device routines ***************************
00060 INTEGER PLST                      ! Declare integer
00070 DIM PLST(4) : MAT READ PLST       ! Dimension, read array
00080 DATA 11,50,16,1
00090 DIM NLST$(1) : NLST$(1) = ' '     ! Dimension, assign value
00100 CALL GDDM ('DSOPEN',2,1,'6180    ',4,PLST(),0,NLST$())
00110 ! Open plotter device 2 of family 1 named 6180,
00120 ! using PLST option group 11 value 50 (pen speed 50% of max),
00130 ! and using group 16 option 1 (horizontal paper orientation);
00140 ! name list has 0 names in array NLST$
00150 CALL GDDM ('DSUSE',1,2)
00160 ! Use device 2 as active device (option 1)
00170 ! ****************** Symbol set ****************************
00180 CALL GDDM ('GSLSS',2,'ADMUWTRP',66)
00190 ! Load vector symbol set ADMUWTRP as symbol set #66
00200 ! **********  DEFINE THE CHART LAYOUT **********************
00210 ! ****************** Heading *******************************
00220 INTEGER HATT                      ! Declare heading attribute
00230 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00240 DATA 2,3,66,200
00250 ! 2 = pen 2, 3 = char mode, 66 = symbol set number, 200 = size
00260 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00270 CALL GDDM ('CHHEAD',11,'TOTAL SALES')
00280 ! Write 11-character heading with 'character string'
```

```
00290 ! ***************** Axis **********************************
00300 CALL GDDM ('CHXTTL',5,'MONTH')     ! Write 5-character x title
00310 CALL GDDM ('CHYTTL',23,'UNITS SOLD IN THOUSANDS')    ! y title
00320 CALL GDDM ('CHXMTH',1)             ! Month labels begin with JAN
00330 CALL GDDM ('CHXSET','NOFORCEZERO') ! Label JAN starts at y axis
00340 ! ***************** Legend ********************************
00350 CALL GDDM ('CHSET','KBOX')         ! Put legend inside box.
00360 CALL GDDM ('CHKEYP','H','B','C')
00370 ! Legend position horizontal, bottom, centered
00380 CALL GDDM ('CHKEY',2,7,'Group AGroup B')
00390 ! Write two 7-character legend key labels using 'string'
00400 ! ********* Set attributes for Line Chart *****************
00410 INTEGER VATT                           ! Declare value attributes
00420 DIM VATT(6) : MAT READ VATT            ! Dimension, read array
00430 DATA 3,3,0,100,100,9000
00440 ! 3 = pen 3, 3 = char mode, 0 = default symbol set
00450 ! 100 = default char size, 100 = default height/width multiplier
00460 ! 9000 = 90 degree rotation
00470 CALL GDDM('CHSET','VALUES')            ! Draw value labels
00480 CALL GDDM('CHVATT',6,VATT())           ! Set value label attributes
00490 CALL GDDM ('CHSET','CURVE')            ! Smooth the plotted lines
00500 ! ********* Specify data for Line Chart ******************
00510 DIM MONTHS(12) : MAT READ MONTHS       ! Array for x axis
00520 DATA  1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12
00530 DIM SALES(24) : MAT READ SALES         ! Array for y axis
00540 DATA 5,6,8,11,14,19,13,10,11,9,7,6
00550 DATA 13,12,14,19,20,27,25,17,17,14,13,12
00560 ! *************** DRAW LINE CHART **************************
00570 CALL GDDM ('CHPLOT',2,12,MONTHS(),SALES())
00580 ! Draw chart with 2 components (lines), 12 data points per line
00590 ! *************** SEND TO DEVICE ***************************
00600 CALL GDDM ('FSFRCE')                   ! Send to device
00610 ! ***************** TERMINATE *****************************
00620 CALL GDDM ('FSTERM')                   ! Terminate graphics
00630 END                                    ! End BASIC program
```

Drawing Scatter Plots

Setting the Color Selection Order

CHCOL – Set component color. CHCOL sets the color of the markers, by defining a table that holds the number of colors and the order of their selection.

If CHCOL is not specified, the sequence of colors in the default color table is used. The default color table is the GDDM color table which is either the default color table for the current page or a color table modified for use in the current page.

Setting the Marker Type Selection Order

CHMARK – Set component marker. CHMARK sets the type of marker used by each of the components, by defining a table that holds the number of markers and the order of their selection.

If CHMARK is not specified, the sequence of markers in the default marker table is used. The default marker table shown for the discussion of the GSMRKS routine in Chapter 3, "Using GDDM" shows the order of selection.

Writing Data Values

CHSET – Specify chart options. CHSET(NOVALUES|VALUES) controls how values are displayed. CHSET(VALUES) writes the data value represented by each data point adjacent to the data point. CHSET(NOVALUES) indicates that no data values are displayed for the data points.

CHSET(BVALUES) blanks the areas where data values are written. When the area is blanked, no other display feature can occupy the data value text box. When the area is not blanked (NBVALUES), the values can overpaint the component.

CHVCHR – Number of data value characters. CHVCHR sets the number of characters to be used for showing data values on a scatter plot. The value text attributes are controlled by CHVATT.

Up to 15 characters can be used, but some of the 15 characters from one data value would probably overwrite those for the next data point.

By default, the true value represented by the data point is used, up to 9 character positions.

CHSET – Specify chart options. CHSET(NPGFS|PGFS) specifies the method of punctuating numbers greater than 1000 displayed on the chart. PGFS suppresses the punctuation except for the decimal point. NPGFS uses OS/400 system value QDECFMT to specify the type of punctuation used for numbers greater than 1000 by your system.

CHVATT – Value text attributes. CHVATT specifies the attributes for value text. Value text is used to show individual values for data points on a scatter plot. Attributes that can be set are color, and character font, size, and rotation.

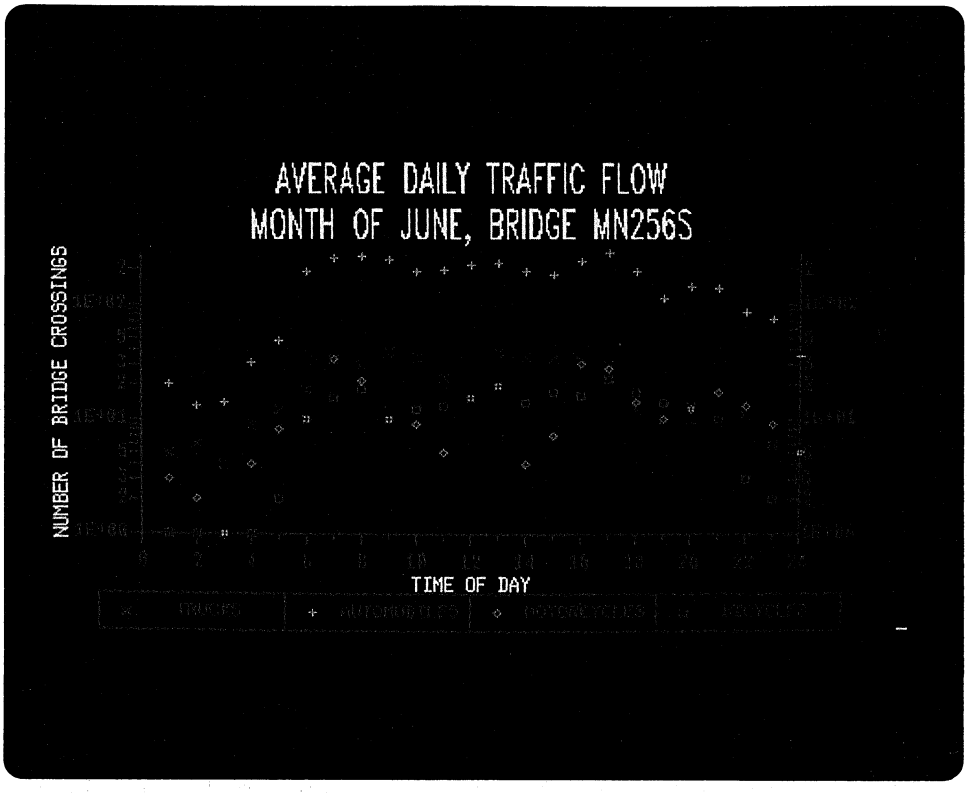
If CHVATT is not specified, each data value is shown in standard-size device default characters, in the default color, in the default 0 rotation.

Drawing the Scatter Plot

CHSET – Specify chart options. CHSET (NOLINES) suppresses the interconnecting lines between points of a line chart, resulting in scatter plot.

CHPLOT – Draw a line graph or a scatter plot. CHPLOT draws the scatter plot. Parameters in the CHPLOT routine specify the number of components to be drawn, the number of data points in each component, and the arrays used for the data groups the components are drawn to represent.

The next chart is a scatter plot produced on the display. For this chart, a logarithmic scale is used for the y axis, and the axis is duplicated.




```
00010 ! ***************** TRAFFIC FLOW ***************************
00020 ! ****************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ****************** Symbol set ****************************
00060 CALL GDDM ('GSLSS',2,'ADMUWDRP',66)
00070 ! Load vector symbol set ADMUWDRP as symbol set #66
00080 ! **********  DEFINE THE CHART LAYOUT **********************
00090 ! ****************** Heading *******************************
00100 INTEGER HATT                      ! Declare head attribute array
00110 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00120 DATA 6,3,66,225
00130 ! 6 = yellow, 3 = char mode, 66 = symbol set number, 225 = size
00140 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00150 DIM CHAR$*60                      ! Dimension char var to 60
00160 CHAR$ = 'AVERAGE DAILY TRAFFIC FLOW;MONTH OF JUNE, BRIDGE MN256S'
00170 CALL GDDM ('CHHEAD',55,CHAR$)
00180 ! Write 55-character heading with 'character string'
00190 ! ****************** Axes **********************************
00200 INTEGER TATT                      ! Declare title attribute array
00210 DIM TATT(4) : MAT READ TATT       ! Dimension, read array
00220 DATA 7,3,0,100
00230 ! 7 = white, 3 = char mode, 0 = default symbol set, 100 = size
00240 CALL GDDM ('CHTATT',4,TATT())     ! Set axis title attributes
00250 CHAR$ = 'TIME OF DAY'
00260 CALL GDDM ('CHXTTL',11,CHAR$)     ! Write 11-character y-title
00270 CHAR$ = 'NUMBER OF BRIDGE CROSSINGS'
00280 CALL GDDM ('CHYTTL',26,CHAR$)     ! Write 26-character x-title
00290 CALL GDDM ('CHYSET','LOGARITHMIC')! Use log scale for y axis
00300 CALL GDDM ('CHSET','YDUP')        ! Draw duplicate y axis
00310 CALL GDDM ('CHYTIC',10.0,1.0)     ! Add major ticks for log axis
00320 CALL GDDM ('CHXTIC',2.0,1.0)      ! Add minor ticks for x axis
00330 CALL GDDM ('CHSET','NDRAW')       ! Suppress axes until CHDRAX
```

```
00340 ! ***************** Legend ********************************
00350 CALL GDDM ('CHKEYP','H','B','C')  ! Position legend
00360 ! 'H' = horizontal, 'B' = bottom, 'C' = centered
00370 CHAR$ = '  TRUCKS   AUTOMOBILESMOTORCYCLES BICYCLES  '
00380 CALL GDDM ('CHKEY',4,11,CHAR$)
00390 ! Write four 11-character legend key labels using 'string'
00400 CALL GDDM ('CHSET','KBOX')        ! Enclose legend in box
00410 ! ********* Specify data for Scatter Plot *****************
00420 DIM TIMES(24) : MAT READ TIMES          ! Array for x axis
00430 DATA   1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12
00440 DATA  13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
00450 DIM TRAFFIC(96) : MAT READ TRAFFIC      ! Array for y axis
00460 DATA   5,  6,  4,  9, 12, 18, 33, 30, 37, 34, 23, 15
00470 DATA  37, 34, 33, 32, 30, 12, 10, 10,  9, 11,  6,  4
00480 DATA  20, 13, 14, 31, 48,189,240,254,237,184,195,215
00490 DATA 219,184,173,230,266,189,110,140,132, 84, 74, 35
00500 DATA   3,  2,  1,  4,  8, 10, 33, 21, 10,  9,  5, 15
00510 DATA  19,  4,  7, 30, 27, 14, 10, 12, 17, 13,  9,  5
00520 DATA   1,  1,  1,  1,  2, 10, 15, 18, 10, 12, 13, 15
00530 DATA  19, 14, 17, 16, 22, 17, 14, 13, 10,  3,  2,  3
00540 ! *************** DRAW SCATTER PLOT ***********************
00550 CALL GDDM ('CHSET','NOLINES')             ! Suppress lines
00560 CALL GDDM ('CHPLOT',4,24,TIMES(),TRAFFIC())! Draw scatter plot
00570 CALL GDDM ('CHDRAX')                      ! Draw axes
00580 ! *************** Send to display *************************
00590 INTEGER ATTYPE,ATMOD,COUNT
00600 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)    ! Send to display
00610 ! ***************** TERMINATE ******************************
00620 CALL GDDM ('FSTERM')                       ! Terminate graphics
00630 END                                        ! End BASIC program
```

Drawing Surface Charts

Setting the Component Color Selection Order

CHCOL – Set component color. CHCOL sets the color of the components, by defining a table that holds the number of colors and the order of their selection.

If CHCOL is not specified, the sequence of colors in the default color table is used. The default color table is the GDDM color table which is either the default color table for the current page or a color table modified for use in the current page.

Setting the Line Curve

CHSET – Specify chart options. CHSET (CURVE) draws the components with line curving; the degree of line curving is specified by the CHFINE routine. If CHSET(CURVE) is specified, the default value of CHFINE is 10.

A curved line is more attractive than one with straight lines connecting data points, but the curved line can be misleading on a chart where a high degree of accuracy is needed. Presentation Graphics passes the curved line *through* the data points, but does not interpolate the true values that lie between data points; therefore, values on the curved line between data points can be inaccurate.

CHFINE – Set line curving smoothness. CHFINE sets the degree of curve smoothness of the components.

By default, the degree of smoothness is 10. A number higher than 10 might not add any noticeable degree of smoothness to the curve but will substantially increase the time needed by the system to process and draw the chart.

Writing Data Values

CHSET – Specify chart options. CHSET(NOVALUES|VALUES) controls how values are displayed. CHSET(VALUES) writes the data value represented by each data point adjacent to the data point. CHSET(NOVALUES) indicates that no data values are displayed for the data points.

CHSET(BVALUES) blanks the areas where data values are written. When the area is blanked, no other display feature can occupy the data value text box. When the area is not blanked (NBVALUES), the values can overpaint the component.

CHVCHR – Number of data value characters. CHVCHR sets the number of characters to be used for showing data values on a surface chart. The value text attributes are controlled by CHVATT.

Up to 15 characters can be used, but some of the 15 characters from one data value would probably overwrite those for the next data point.

By default, the true value represented by the data point is used, up to 9 character positions.

CHSET – Specify chart options. CHSET(NPGFS|PGFS) specifies the method of punctuating numbers greater than 1000 displayed on the chart. PGFS suppresses the punctuation except for the decimal point. NPGFS uses OS/400 system value QDECFMT to specify the type of punctuation used for numbers greater than 1000 by your system.

CHVATT – Value text attributes. CHVATT specifies the attributes for value text. Value text is used to show individual values for data points on a surface chart. Attributes that can be set are color, and character font, size, and rotation.

If CHVATT is not specified, each data value is shown in standard-size device default characters, in the default color, in the default 0 rotation.

Setting the Shading Attributes

CHPAT – Set component shading pattern. CHPAT sets the type of pattern used by shaded components, by defining a table that holds the number of patterns and the order of their selection.

If CHPAT is not specified, the sequence of patterns in the default pattern table is used. The default pattern table shown for the discussion of the GSPAT routine in Chapter 3, "Using GDDM" shows the order of selection.

CHSET – Specify chart options. CHSET (INFILL) suppresses the shading of the first component, which results in a *floating surface chart*. Floating surface charts are described on page 4-60.

CHSET (NOFILL) suppresses the shading of all the components, which results in a line chart.

Setting the Type of Shading to be Performed

CHSET — Specify chart options. CHSET(NOMOUNTAIN|MOUNTAIN) specifies whether mountain range shading or normal shading is to be performed.

Normal shading (CHSET(NOMOUNTAIN)) is performed in the following manner:

- The first component is shaded from the x-axis (or state 1 y datum line, if any). This line is called the reference line or the shading reference line.
- Every other component is shaded from the previous component's data line.

In cases in which the components overlap one another, a mixing mode of 'OR' is used.

Mountain range shading (CHSET(MOUNTAIN)) is performed in the following manner:

- The last component is shaded from the x-axis (or state 1 y datum line, if any).
- Every other component is shaded, in reverse order, also from the x-axis (or y datum line, if any), and NOT from the previous component's data line, as in the case of normal shading.

Mountain range shading always uses the default mix mode of overpaint, except for hardcopy devices, which use a modified mix mode (color is the result of etching one color over the top of another).

If relative data is used (CHSET(RELATIVE)), there will be no difference between mountain range shading and normal shading.

Setting the Type of Data to be Shown

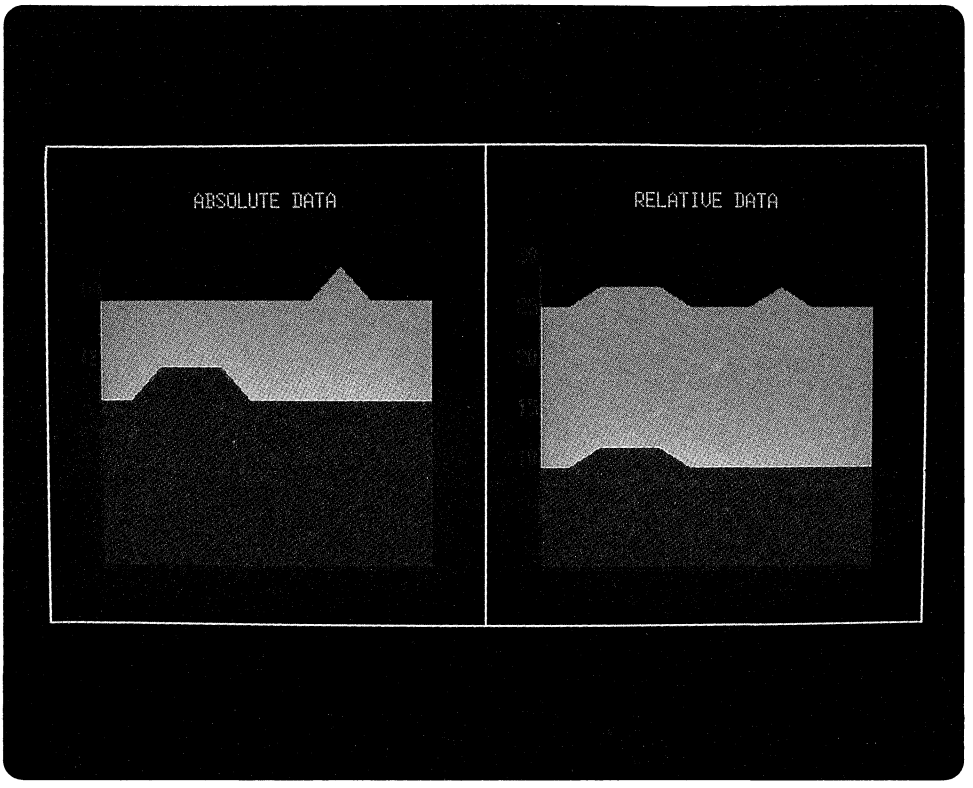
CHSET — Specify chart options. CHSET (**ABSOLUTE**) shows the data as absolute data.

CHSET (RELATIVE) shows the data as relative data.

Data for surface charts can be shown as either *absolute* data or as *relative* data.

Absolute data (the default) is shown where the upper boundary of the shaded region shows the true values of the y-data for that component.

Relative data is shown where the upper boundary of a component shows the sum of its own y-values added to those of the components shown below it. The true y-value of the component can be seen be comparing the thickness of its shaded region with the y-axis scale.

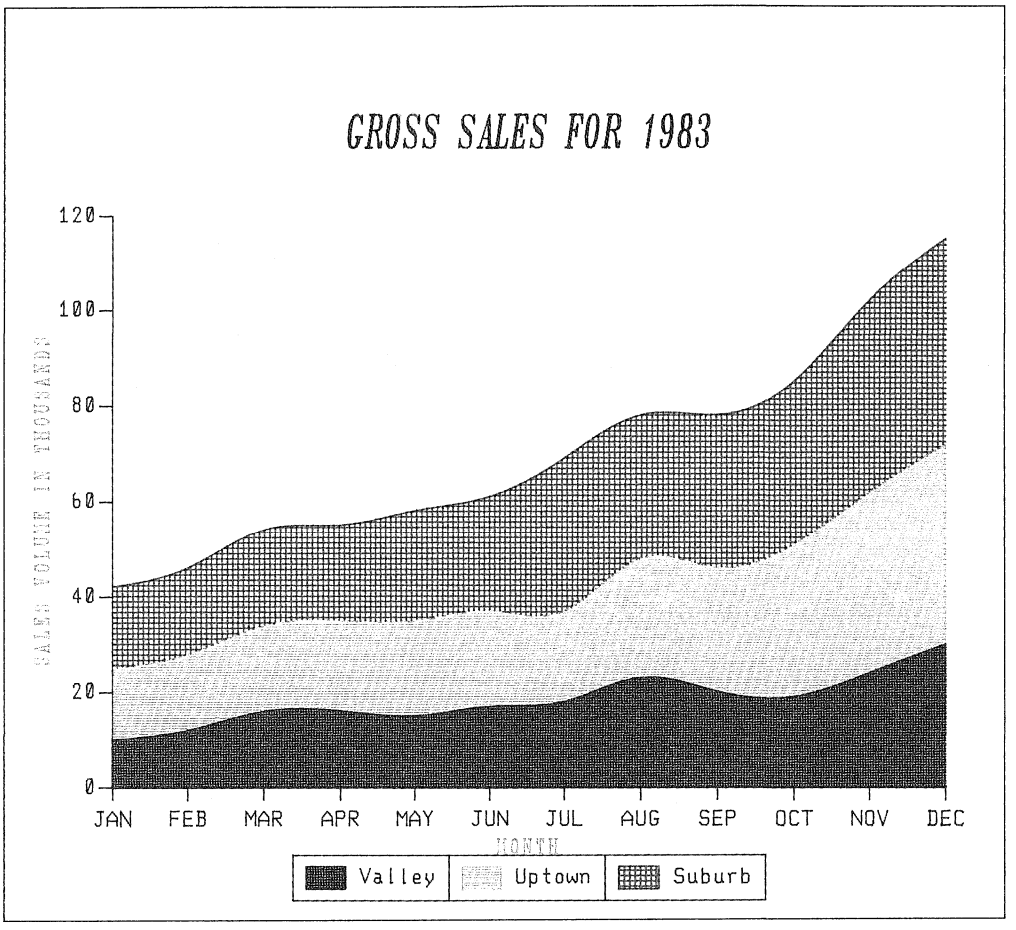


Absolute versus relative data. The picture shows two surface charts whose data groups are identical, but the method of showing the data differs. Note the difference in the scale range used for the two charts.

Drawing the Surface Chart

CHSURF – Drawing a surface chart. CHSURF draws a surface chart. Parameters in the CHSURF routine specify the number of components to be drawn, the number of data points in each component, and the arrays used for the data groups the components are drawn to represent.

This is a surface chart produced on a plotter. It shows relative data, where the data values are stacked upon one another. The chart also uses month labels for the x axis.




```
00010 ! ***************** STORE SALES ***************************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ************** Device routines ***************************
00060 INTEGER PLST                      ! Declare integer
00070 DIM PLST(4) : MAT READ PLST       ! Dimension, read array
00080 DATA 11,50,16,1
00090 DIM NLST$(1) : NLST$(1) = ' '     ! Dimension, assign value
00100 CALL GDDM ('DSOPEN',2,1,'6180    ',4,PLST(),0,NLST$())
00110 ! Open plotter device 2 of family 1 named 6180,
00120 ! using PLST option group 11 value 50 (pen speed 50% of max),
00130 ! and using group 16 option 1 (horizontal paper orientation);
00140 ! name list has 0 names in array NLST$
00150 CALL GDDM ('DSUSE',1,2)
00160 ! Use device 2 as active device (option 1)
```

```
00170 ! ***************** Symbol set ****************************
00180 CALL GDDM ('GSLSS',2,'ADMUWTIP',66)
00190 ! Load vector symbol set ADMUWTIP as symbol set #66
00200 CALL GDDM ('GSLSS',2,'ADMUVCRP',67)
00210 ! Load vector symbol set ADMUVCRP as symbol set #67
00220 ! **********  DEFINE THE CHART LAYOUT ********************
00230 ! ***************** Heading *******************************
00240 INTEGER HATT                      ! Declare heading attribute
00250 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00260 DATA 5,3,66,225
00270 ! 5 = pen 5, 3 = char mode, 66 = symbol set number, 225 = size
00280 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00290 CALL GDDM ('CHHEAD',20,'GROSS SALES FOR 1983')
00300 ! Write 20-character heading with 'character string'
00310 ! ******************** Axes *******************************
00320 INTEGER TATT                      ! Declare axis title attribute
00330 DIM TATT(4) : MAT READ TATT       ! Dimension, read array
00340 DATA 2,3,67,100
00350 ! 2 = pen 2, 3 = char mode, 67 = symbol set number, 100 = size
00360 CALL GDDM ('CHTATT',4,TATT())     ! Set title attributes
00370 CALL GDDM ('CHXTTL',5,'MONTH')    ! Write 5-character x-title
00380 CALL GDDM ('CHYTTL',25,'SALES VOLUME IN THOUSANDS') ! y-title
00390 CALL GDDM ('CHXMTH',1)            ! Month labels begin with JAN
00400 CALL GDDM ('CHXSET','NOFORCEZERO') ! Label JAN starts at y axis
00410 ! ******************** Legend *****************************
00420 CALL GDDM ('CHSET','KBOX')        ! Enclose legend in box
00430 CALL GDDM ('CHKEYP','H','B','C')
00440 ! Position legend H = horizontal, B = bottom, C = centered
00450 CALL GDDM ('CHKEY',3,6,'ValleyUptownSuburb')
00460 ! Write three 6-character legend key labels using 'string'
00470 ! ********* Set attributes for Surface Chart ***************
00480 INTEGER VATT                          ! Declare value attributes
00490 DIM VATT(6) : MAT READ VATT           ! Dimension, read array
00500 DATA 1,3,0,100,100,0
00510 ! 1 = pen 1, 3 = char mode, 0 = default symbol set
00520 ! 100 = default char size, 100 = default height/width multiplier
00530 ! 0 = default rotation (0 degrees)
00540 CALL GDDM('CHSET','VALUES')           ! Draw value labels
00550 CALL GDDM('CHVATT',6,VATT())          ! Set value label attributes
00560 CALL GDDM ('CHSET','CURVE')           ! Smooth the plotted lines
00570 ! ********* Specify data for Surface Chart *****************
00580 DIM MONTHS(12) : MAT READ MONTHS
00590 DATA 1,2,3,4,5,6,7,8,9,10,11,12
00600 DIM SALES(36) : MAT READ SALES
00610 DATA 10,12,16,16,15,17,18,23,20,19,24,30
00620 DATA 15,16,18,19,20,20,19,25,26,32,38,42
00630 DATA 17,18,20,20,23,24,32,30,32,34,40,43
00640 CALL GDDM ('CHSET','RELATIVE')    ! Show relative data
00650 ! *************** DRAW SURFACE CHART ***********************
00660 CALL GDDM ('CHSURF',3,12,MONTHS(),SALES())
00670 ! Draw chart with 3 components (areas), 12 data points per line
00680 ! *************** Send to display **************************
00690 CALL GDDM ('FSFRCE')                      ! Send to display
00700 ! ***************** TERMINATE *****************************
00710 CALL GDDM ('FSTERM')                      ! Terminate graphics
00720 END                                       ! End BASIC program
```

Drawing a Floating Surface Chart

A floating surface chart can be useful for showing ranges of values.

For a floating surface chart, the lower boundary line for the lowest region is defined with *dummy* values that define the boundary but do not represent a filled area. In the program, these dummy values are shown in the data array as the first component.

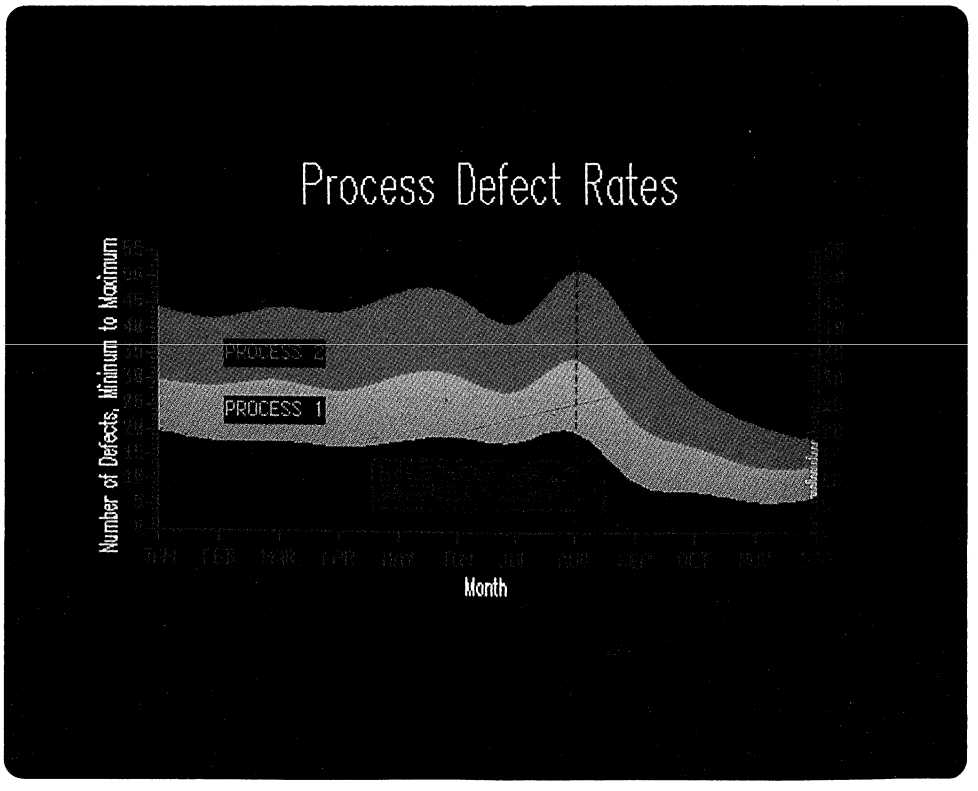
To draw a floating surface chart, use the routines for a surface chart plus the following routines:

CHSET – Specify chart options. CHSET (RELATIVE) shows the data as relative data. A floating surface chart should be used with relative data specified so that the widths of the components are significant.

CHSET – Specify chart options. CHSET (INFILL) suppresses the shading of the first component, which creates a floating surface chart.

The following chart is a floating surface chart that uses CHSET (INFILL) to suppress the shading of the first component. The data for the first component is used to set the lower limit of the range represented by component 2 (labeled PROCESS 1).

The program sets both fill patterns to solid and specifies a scale that uses a major tick mark every five units for the y axis (which is duplicated) and four minor tick marks between each major tick mark. Also, no legend is used; instead, chart notes are used to label the components because, if a legend is used, the first component is represented in the legend even though its shading is suppressed. A datum line and similarly-colored note is used to help explain the chart. The chart is produced on the display:



```
00010 ! ****************** DEFECT RATE ***********************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ***************** Symbol set ****************************
00060 CALL GDDM ('GSLSS',2,'ADMUWSRP',66)
00080 ! **********  DEFINE THE CHART LAYOUT *********************
00090 ! ***************** Heading *******************************
00100 INTEGER HATT                      ! Declare heading attribute
00110 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00120 DATA 7,3,66,300
00130 ! 7 = white, 3 = char mode, 66 = symbol set number, 300 = size
00140 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00150 CALL GDDM ('CHHEAD',20,'Process Defect Rates')
00160 ! Write 20-character heading with 'character string'
00170 INTEGER PAT                       ! Declare pattern array
00180 DIM PAT(2) : MAT READ PAT         ! Dimension, read array
00190 DATA 16,16
00200 CALL GDDM ('CHPAT',2,PAT())       ! Set pattern attributes
00210 INTEGER COL : DIM COL(3) : MAT READ COL   ! Colors array
00220 DATA 2,2,3
00230 CALL GDDM ('CHCOL',3,COL())        ! Set color attributes
00240 ! ****************** Axes *********************************
00250 INTEGER TATT                       ! Declare axis title attribute
00260 DIM TATT(4) : MAT READ TATT        ! Dimension, read array
00270 DATA 7,3,66,125
00280 ! 7 = white, 3 = char mode, 66 = symbol set number, 125 = size
00290 CALL GDDM ('CHTATT',4,TATT())      ! Set title attributes
00300 CALL GDDM ('CHXTTL',5,'Month')     ! Write 5-character x-title
00310 DIM CHAR$*50                       ! 50 characters maximum
00320 CHAR$ = 'Number of Defects, Minimum to Maximum'
00330 CALL GDDM ('CHYTTL',38,CHAR$)      ! Write y-title
00340 CALL GDDM ('CHXMTH',1)             ! Month labels begin with JAN
00350 CALL GDDM ('CHXSET','NOFORCEZERO') ! Label JAN starts at y axis
00360 CALL GDDM ('CHSET','YDUP')         ! Duplicate y axis
00370 CALL GDDM ('CHYTIC',5.0,4.0)       ! Add minor tick marks
00380 ! ********* Set attributes for Surface Chart **************
00390 CALL GDDM ('CHSET','INFILL')           ! Suppress 1st area
00400 CALL GDDM ('CHSET','CURVE')            ! Smooth the plotted lines
00410 ! ********* Specify data for Surface Chart ****************
00420 DIM MONTHS(12) : MAT READ MONTHS       ! Array for x axis
00430 DATA  1, 2, 3, 4, 5, 6, 7, 8, 9,10,11,12
00440 DIM DEFECTS(36) : MAT READ DEFECTS     ! Array for y axis
00450 DATA 20,18,18,17,18,19,18,20,10, 8, 6, 7
00460 DATA 10,11,12,11,13,12,10,14,12, 9, 7, 6
00470 DATA 14,13,14,15,16,15,13,17,17,10, 8, 5
00480 CALL GDDM ('CHSET','RELATIVE')         ! Show relative data
00490 ! *************** DRAW SURFACE CHART ***********************
00500 CALL GDDM ('CHSURF',3,12,MONTHS(),DEFECTS())
00510 ! Draw chart with 3 components (areas), 12 data points per line
00520 CALL GDDM ('CHDRAX')               ! Draw axes again
00530 INTEGER DATT                       ! Declare datum line attribute
00540 DIM DATT(2) : MAT READ DATT        ! Dimension, read array
00550 DATA 1,5
00560 ! 1 = blue, 5 = dashed             ! Datum line attributes
00570 ! ************* Draw datum line ****************************
00580 CALL GDDM ('CHDATT',2,DATT())      ! Set datum attributes
00590 CALL GDDM ('CHXDTM',8.0)           ! Draw datum line from X
00600 ! ************* Write chart notes **************************
00610 INTEGER NATT                       ! Declare integer variable
```

```
00620 DIM NATT(2) : MAT READ NATT       ! Dimension, read array
00630 DATA 1,2
00640 ! 1 = blue, 2 = char mode         ! Note attributes
00650 CALL GDDM ('CHNATT',2,NATT())     ! Set note attributes
00660 CALL GDDM ('CHSET','NBOX')        ! Enclose note in box
00670 CALL GDDM ('CHSET','BNOTE')       ! Blank note area
00680 CALL GDDM ('CHNOFF',4.5,3.5)      ! Position note
00690 CHAR$ = 'IMPLEMENTED QUALITY;AWARENESS PROGRAM 8/1'
00700 CALL GDDM ('CHNOTE','Z7',41,CHAR$)! Write note
00710 NATT(1) = 2                       ! Note color red
00720 CALL GDDM ('CHNATT',2,NATT())     ! Set note attributes
00730 CALL GDDM ('CHNOFF',2.0,20.0)     ! Position note
00740 CALL GDDM ('CHNOTE','Z7',9,'PROCESS 1')
00750 NATT(1) = 3                       ! Note color pink
00760 CALL GDDM ('CHNATT',2,NATT())     ! Set note attributes
00770 CALL GDDM ('CHNOFF',2.0,31.0)     ! Position note
00780 CALL GDDM ('CHNOTE','Z7',9,'PROCESS 2')    ! Write note
00790 ! *************** Send to display *************************
00800 INTEGER ATTYPE,ATMOD,COUNT                 ! Declare integers
00810 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)    ! Send to display
00830 CALL GDDM ('FSTERM')                       ! Terminate graphics
00840 END                                        ! End BASIC program
```

Drawing Bar Charts

Setting the Component Color Selection Order

CHCOL – Set component color. CHCOL sets the color of the components, by defining a table that holds the number of colors and the order of their selection.

If CHCOL is not specified, the sequence of colors in the default color table is used. The default color table is, in this case, the GDDM color table which is either the default color table for the current page or a color table modified for use in the current page.

Setting the Bar Attributes

CHPAT – Set component shading pattern. CHPAT sets the type of pattern used by shaded components, by defining a table that holds the number of patterns and the order of their selection.

If CHPAT is not specified, the sequence of patterns in the default pattern table is used. The default pattern table shown for the discussion of the GSPAT routine in Chapter 3, "Using GDDM" shows the order of selection.

CHSET – Specify chart options. CHSET (NOFILL) suppresses the shading of bars.

Writing Bar Values

CHSET – Specify chart options. CHSET (**NOVALUES**|VALUES|CVALUES) controls how values are displayed. CHSET(VALUES) writes the data value represented by each bar adjacent to the bar. CHSET(NOVALUES) indicates that no data values are displayed on the bars.

CHSET(CVALUES) controls how values are displayed on the bars. Use it followed by CHSET(VINSIDE|**VONTOP**) to display the values inside (VINSIDE) or above (VONTOP) the bars.

CHSET (BVALUES) blanks the areas where bar values are written. When the area is blanked, no other display feature can occupy the bar value text box. When the area is not blanked (NBVALUES), the values can *overpaint* the component.

CHVCHR – Number of data value characters. CHVCHR sets the number of characters to be used for showing bar values on a bar chart. The value text attributes are controlled by CHVATT.

Up to 15 characters can be used, but some of the 15 characters from one bar value would probably overwrite those for the next bar (for character-mode 2), or would be too small to read (for character-mode 3).

By default, the true value represented by the bar is used, up to 9 character positions. The picture of the single-bar chart shown later has bar values 2 characters long.

CHSET – Specify chart options. CHSET (**NPGFS**|PGFS) specifies the method of punctuating numbers greater than 1000 displayed on the chart. PGFS suppresses the punctuation except for the decimal point. NPGFS uses OS/400 system value QDECFMT to specify the type of punctuation used for numbers greater than 1000 by your system.

CHVATT – Value text attributes. CHVATT specifies the attributes for value text. Value text is used to show individual values for bars on a bar chart. Attributes that can be set are color, and character font, size, and rotation.

If CHVATT is not specified, each bar value is shown in standard-size device default characters, in the default color, with the default rotation.

Setting the Bar Spacing

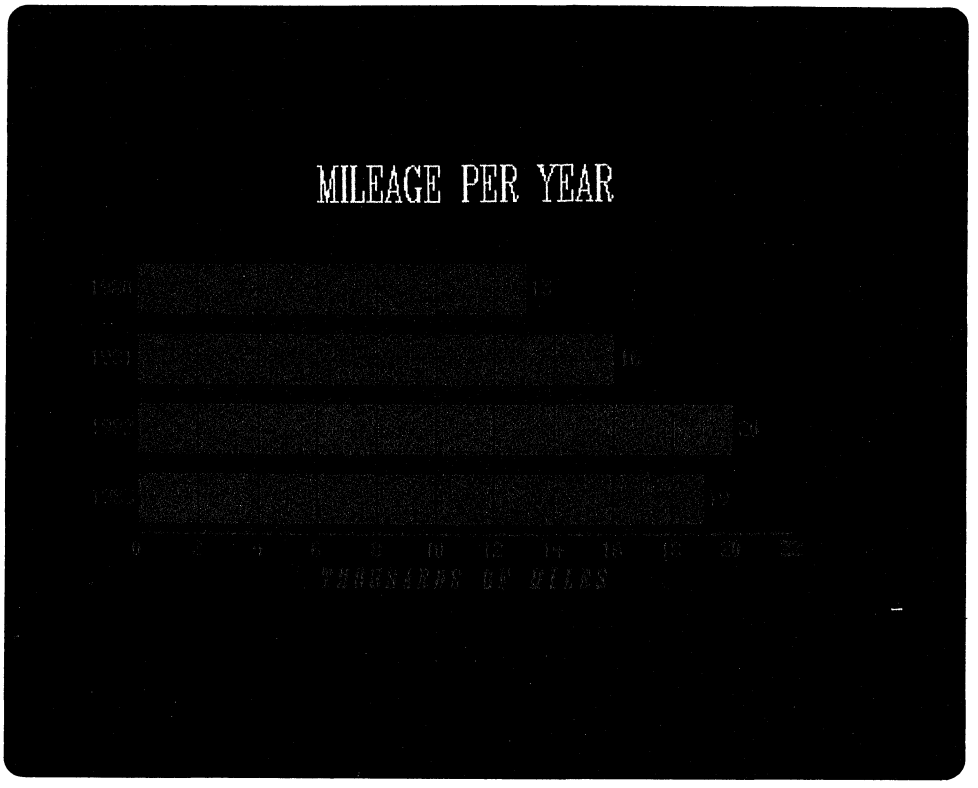
CHGAP – Spacing between bars. CHGAP sets the distance between the bars in a bar chart. By default, the distance is one-half the width of each individual bar. A negative value gives overlapping bars.

Drawing the Bar Chart

CHBAR – Draw a bar chart. CHBAR draws a bar chart. Parameters in the CHBAR routine specify the number of components to be drawn, the number of bars in each component, and the array used for the dependent variables (the heights of the bars).

The following chart is an example of a single-bar chart produced on the display. The chart is oriented horizontally, and it uses an *invisible* grid to suppress the x axis (which is vertical) and to overpaint the chart components to increase the usability of

the chart. The tick marks for the axes are suppressed to enhance the simple appearance of the chart.




```
00010 ! ****************** MILEAGE ******************************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ***************** Symbol set ****************************
00060 CALL GDDM ('GSLSS',2,'ADMUWCRP',66)
00070 ! Load vector symbol set ADMUWCRP as symbol set #66
00080 CALL GDDM ('GSLSS',2,'ADMUVTIP',67)
00090 ! Load vector symbol set ADMUVTIP as symbol set #67
00100 ! **********  DEFINE THE CHART LAYOUT *********************
00110 CALL GDDM ('CHSET','XVERTICAL')   ! Horizontal format
00120 INTEGER HATT                      ! Declare heading attribute
00130 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00140 DATA 6,3,66,275
00150 ! 6 = yellow, 3 = char mode, 66 = symbol set number, 275 = size
00160 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00170 CALL GDDM ('CHHEAD',16,'MILEAGE PER YEAR')
00180 ! Write 16-character heading with 'character string'
00190 ! ****************** Axes *********************************
00200 CALL GDDM ('CHSET','NDRAW')   ! Suppress axes/grid until CHDRAX
00210 CALL GDDM ('CHXSET','PLAIN')  ! Suppress x-axis tick marks
00220 CALL GDDM ('CHYSET','PLAIN')  ! Suppress y-axis tick marks
00230 CALL GDDM ('CHYRNG',0.0,22.0) ! Set specific range for y axis
00240 INTEGER TATT                      ! Declare axis title attribute
00250 DIM TATT(4) : MAT READ TATT       ! Dimension, read array
00260 DATA 4,3,67,150
00270 ! 4 = green, 3 = char mode, 67 = symbol set number, 150 = size
00280 CALL GDDM ('CHTATT',4,TATT())     ! Set title attributes
00290 CALL GDDM ('CHYTTL',18,'THOUSANDS OF MILES')  ! y-axis title
```

```
00300 INTEGER LATT                          ! Declare axis label attribute
00310 DIM LATT(2) : MAT READ LATT           ! Dimension, read array
00320 DATA 4,2
00330 ! 4 = green, 2 = character mode
00340 CALL GDDM ('CHLATT',2,LATT())         ! Set label attributes
00350 CALL GDDM ('CHXLAB',4,4,'1980198119821983')
00360 ! Write 4 4-character x-axis labels with 'character string'
00370 ! ***************** Specify grid **************************
00380 INTEGER GATT                          ! Declare axis grid attribute
00390 DIM GATT(6) : MAT READ GATT           ! Dimension, read array
00400 DATA 0,0,0,8,0,2
00410 ! First three elements are for an x-axis grid (not used here)
00420 ! For y-axis grid, 8 - background, 0 = solid line, 2 = wide line
00430 CALL GDDM ('CHGATT',6,GATT())  ! Set grid attributes
00440 CALL GDDM ('CHYSET','GRID')    ! Draw invisible grid from y axis
00450 ! ****************** Bar values ***************************
00460 INTEGER VATT                          ! Declare bar value attribute
00470 DIM VATT(2) : MAT READ VATT           ! Dimension, read array
00480 DATA 4,2
00490 ! 4 = green, 2 = char mode
00500 CALL GDDM ('CHVATT',2,VATT())         ! Set bar value attributes
00510 CALL GDDM ('CHSET','VALUES')          ! Show bar values
00520 ! *********** Specify data, draw chart *******************
00530 DIM MILES(4) : MAT READ MILES   ! Dimension and read data array
00540 DATA 13,16,20,19
00550 CALL GDDM ('CHBAR',1,4,MILES())
00560 ! Draw chart with 1 data group of 4 bars
00570 CALL GDDM ('CHDRAX')             ! Draw axes and grid
00580 ! *************** Send to display **************************
00590 INTEGER ATTYPE,ATMOD,COUNT
00600 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)    ! Send to display
00610 ! ***************** TERMINATE *****************************
00620 CALL GDDM ('FSTERM')                       ! Terminate graphics
00630 END                                        ! End BASIC program
```

Drawing Multiple-Bar Charts

CHSET – Specify chart options. CHSET (MBAR) draws a multiple-bar chart when CHBAR is called.

CHGGAP – Spacing between bar groups. CHGGAP sets the distance between groups of bars in a multiple-bar chart. By default, the distance is twice the width of each individual bar in a group of bars.

CHNUM – Set number of chart components. CHNUM sets the number of individual bars used for each x-axis value. The individual bars are constructed by each call of the CHBAR routine.

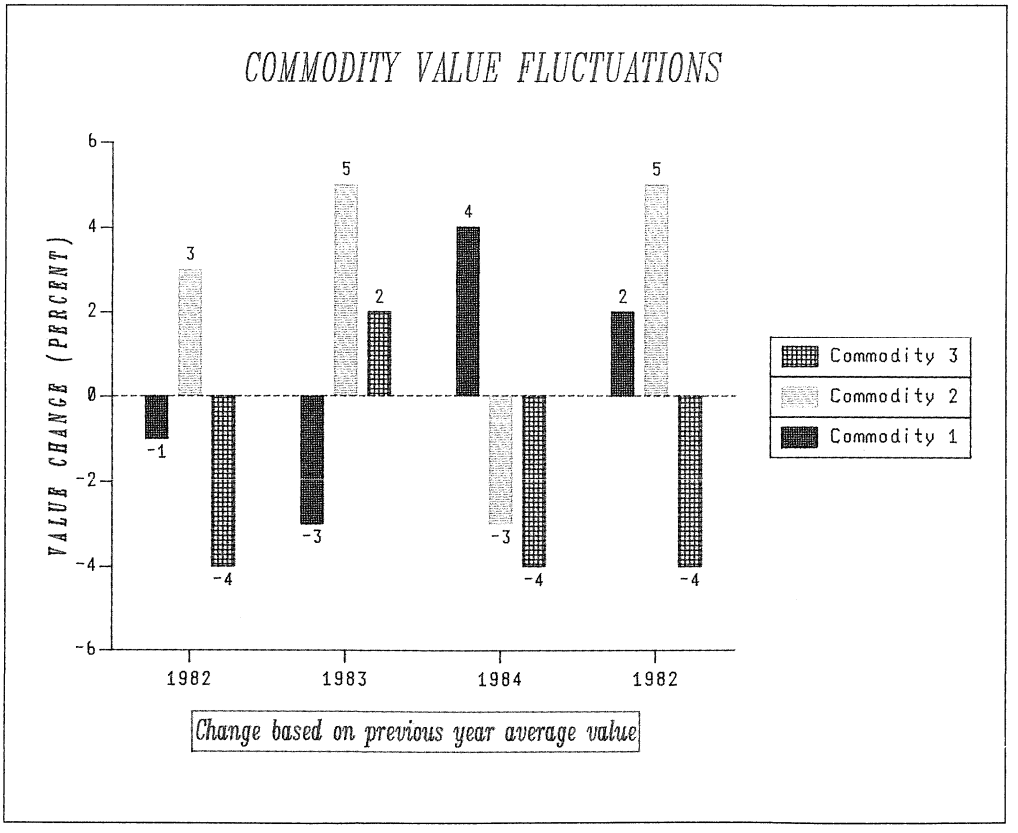
The parameter value passed to Presentation Graphics by CHNUM is the number of bars Presentation Graphics makes room for on the x axis. For example, for a program that draws a chart with two groups of three bars each, CHNUM(3) causes the bars drawn by the first CHBAR routine to be narrower and placed on the x axis so that two other bars can also be drawn by other CHBAR routines.

These combinations of routines draw the same chart as two groups with three bars in each:

1. With one call to CHBAR:
   a. CALL GDDM ('CHBAR',3,2,ARRAY()) where ARRAY has six elements; the first two elements are the values for the first bar in each group, the second two elements are the values for the second bar, and so forth.
2. With two calls to CHBAR:
   a. CALL GDDM ('CHNUM',3).
   b. CALL GDDM ('CHBAR',2,2,ARRAY1()) where ARRAY1 has four elements; the first two elements are the values for first bar in each group, and the second two elements are the values for the second bar.
   c. CALL GDDM ('CHBAR',1,2,ARRAY2()) where ARRAY2 has two elements, which are the values for the last bar in each group.

CHBAR – Draw a bar chart. CHBAR draws a bar chart. Parameters in the CHBAR routine specify the number of components to be drawn, the number of bars in each component of the multiple-bar chart, and the array used for the dependent variables (the heights of the bars).

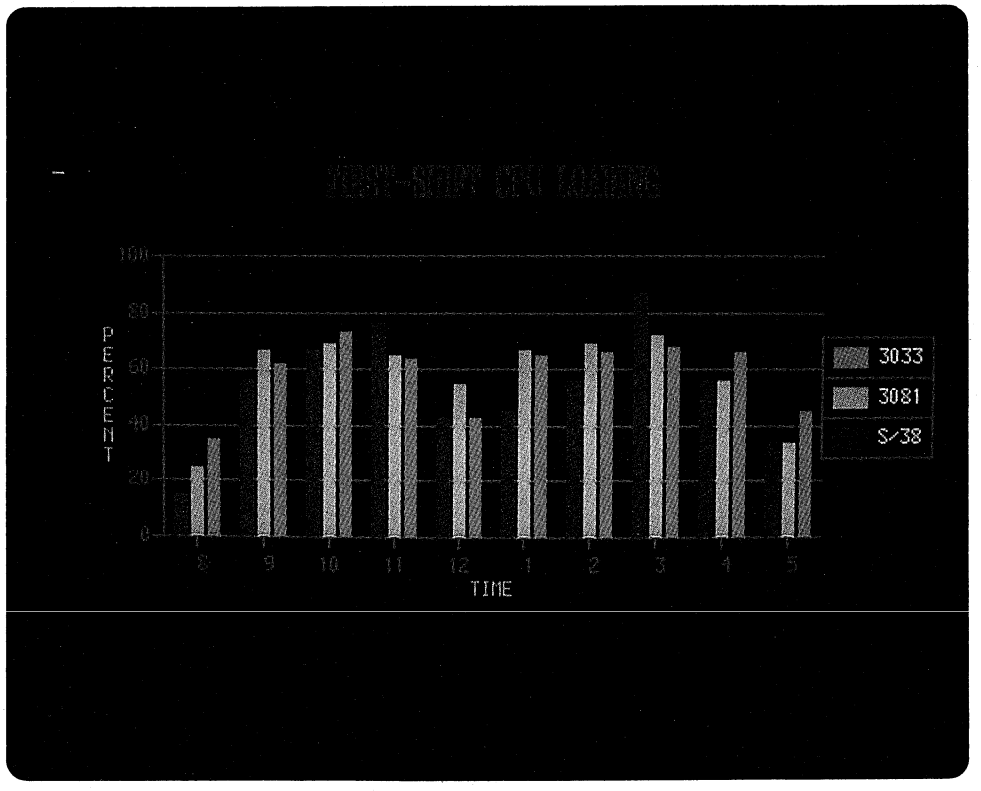
This is an example of a multiple-bar chart produced on a plotter. The chart uses a translated x-axis line to show negative values. The translated axis line is positioned at 0 on the y axis, and the range for the y axis is set from −6 through +6. The gap between the groups of bars is set to 3 (3 times the width of an individual bar).



```
00010 ! ********** COMMODITY - MULTIPLE-BAR ********************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 INTEGER DEVID,FAM,PCT,PLST,NCT,US      ! Declare integers
00060 DEVID=2 : FAM=1 : PCT=4         ! Variables for plotter DSOPEN
00070 ! Device #2, family type 1, 4 items in device parameter list
00080 DIM PLST(4) : MAT READ PLST     ! Declare, read parm list
00090 DATA 11,50,16,1
00100 ! Option #11 has value = 50 for pen speed 50% maximum
00110 ! Option #16 has value = 1 for horizontal paper orientation
00120 NCT = 0                         ! Name count = 0
00130 DIM NLST$(1) : NLST$(1)=' '     ! Name list is empty
00140 CALL GDDM ('DSOPEN',DEVID,FAM,'6180    ',PCT,PLST(),NCT,NLST$())
00150 ! Open device #2, family type 1, 6180 Plotter, using 4-element
00160 ! parm list, name count 0, family name ' '
00170 US=1                            ! Variable for plotter DSUSE
00180 CALL GDDM ('DSUSE',US,DEVID)    ! Use device #2 as current device
00190 ! ***************** Symbol set ****************************
00200 CALL GDDM ('GSLSS',2,'ADMUWCIP',66)
00210 ! Load vector symbol set ADMUWCIP as symbol set #66
00220 CALL GDDM ('GSLSS',2,'ADMUVTIP',67)
00230 ! Load vector symbol set ADMUVTIP as symbol set #67
00240 ! **********  DEFINE THE CHART LAYOUT *********************
00250 INTEGER HATT                      ! Declare heading attribute
00260 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00270 DATA 4,3,66,225
00280 ! 4 = pen 4, 3 = char mode, 66 = symbol set number, 225 = size
00290 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00300 CALL GDDM ('CHHEAD',28,'COMMODITY VALUE FLUCTUATIONS')
00310 ! Write 28-character heading with 'character string'
00320 CALL GDDM ('CHVMAR',10,20)
00330 ! Set right vertical margin to 20 to allow room for legend
00340 ! ******************** Axes *******************************
00350 INTEGER TATT                       ! Declare axis title attribute
00360 DIM TATT(4) : MAT READ TATT        ! Dimension, read array
00370 DATA 4,3,67,125
00380 ! 4 = pen 4, 3 = char mode, 67 = symbol set number, 125 = size
00390 CALL GDDM ('CHTATT',4,TATT())      ! Set title attributes
00400 CALL GDDM ('CHYTTL',22,'VALUE CHANGE (PERCENT)')   ! y-axis title
00410 CALL GDDM ('CHXLAB',3,4,'198219831984')
00420 ! Write 3 4-character x-axis labels with 'character string'
00430 CALL GDDM ('CHYRNG',-6.0,6.0)     ! Set y-axis range
00440 ! ***************** Bar gap *******************************
00450 CALL GDDM ('CHGGAP',3.0)          ! Set bar gap 3*bar width
00460 ! ***************** Bar values ****************************
00470 CALL GDDM ('CHSET','VALUES')      ! Show bar values
00480 ! ****************** Legend *******************************
00490 CALL GDDM ('CHSET','KBOX')        ! Enclose legend in box
00500 DIM CHAR$*50 : CHAR$ = 'Commodity 1Commodity 2Commodity 3'
00510 CALL GDDM ('CHKEY',3,11,CHAR$)  ! 3 11-character legend keys
00520 ! ************ Translated axis line ***********************
00530 INTEGER DATT                      ! Declare line attribute
00540 DIM DATT(2) : MAT READ DATT       ! Dimension, read array
00550 DATA 4,2
00560 ! 4 = pen 4, 2 = short-dashed line
00570 CALL GDDM ('CHDATT',2,DATT())     ! Set axis line attributes
00580 CALL GDDM ('CHYDTM',0.0)          ! Use translated axis at 0
00590 ! ************ Specify data *******************************
00600 DIM DELTA(12) : MAT READ DELTA  ! Dimension and read data array
```

```
00610 DATA  -1,-3, 4, 2, 3, 5,-3, 5,-4, 2,-4,-4
00620 ! ************* Draw chart *******************************
00630 CALL GDDM ('CHBAR',3,4,DELTA())
00640 ! Draw chart with 4 data groups of 3 bars each
00650 ! ************* Write notes ******************************
00660 INTEGER NATT                    ! Declare note attribute
00670 DIM NATT(4) : MAT READ NATT     ! Dimension, read array
00680 DATA 4,3,66,150
00690 ! 4 = pen 4, 3 = mode 3, 66 = symbol set, 150 = size multiplier
00700 CALL GDDM ('CHNATT',4,NATT())   ! Set note attributes
00710 CALL GDDM ('CHSET','NBOX')      ! Enclose note in box
00720 CALL GDDM ('CHNOFF',1.0,1.0)          ! Note position
00730 CHAR$ = 'Change based on previous year average value'
00740 CALL GDDM ('CHNOTE','H1',43,CHAR$)     ! Write note
00750 ! *************** Send to display ************************
00760 CALL GDDM ('FSFRCE')            ! Send to plotter
00770 ! ***************** TERMINATE ******************************
00780 CALL GDDM ('FSTERM')            ! Terminate graphics
00790 END                             ! End BASIC program
```

This multiple-bar chart uses CHNUM to specify that the chart uses three groups of bars:



```
00010 ! ********** CPU LOADS - MULTIPLE-BAR ********************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ***************** Symbol set ****************************
00060 CALL GDDM ('GSLSS',2,'ADMUWTRP',66)
00070 ! Load vector symbol set ADMUWTRP as symbol set #66
00080 ! **********  DEFINE THE CHART LAYOUT **********************
00090 INTEGER HATT                      ! Declare heading attribute
00100 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00110 DATA 5,3,66,225
00120 ! 5 = turq, 3 = char mode, 66 = symbol set number, 225 = size
00130 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00140 CALL GDDM ('CHHEAD',23,'FIRST-SHIFT CPU LOADING')
00150 ! Write 23-character heading with 'character string'
00160 ! ***************** Axes *********************************
00170 INTEGER TATT                      ! Declare axis title attribute
00180 DIM TATT(2) : MAT READ TATT       ! Dimension, read array
00190 DATA 2,2
00200 ! 2 = red, 2 = character mode
00210 CALL GDDM ('CHTATT',2,TATT())      ! Set title attributes
00220 CALL GDDM ('CHXTTL',4,'TIME')      ! x-axis title
00230 CALL GDDM ('CHYTTL',7,'PERCENT')   ! y-axis title
00240 INTEGER LATT                      ! Declare label attribute
00250 DIM LATT(2) : MAT READ LATT       ! Dimension, read array
00260 DATA 4,2
00270 ! 4 = green, 2 = character mode
00280 CALL GDDM ('CHLATT',2,LATT())      ! Set label attributes
00290 CALL GDDM ('CHXLAB',10,2,' 8 9101112 1 2 3 4 5')
00300 ! Write 10 2-character x-axis labels with 'character string'
00310 ! ***************** Grid *********************************
00320 INTEGER GATT                      ! Declare axis grid attribute
00330 DIM GATT(6) : MAT READ GATT       ! Dimension, read array
00340 DATA 0,0,0,4,0,0
00350 ! First three elements for any specified x-axis grid.
00360 ! For y-axis grid, 4 = green, 0 = solid line, 0 = narrow line
00370 CALL GDDM ('CHGATT',6,GATT())      ! Set grid attributes
00380 CALL GDDM ('CHYSET','GRID')        ! Draw grid from y axis
00390 ! ************** Components *********************************
00400 INTEGER PATT                      ! Declare pattern array
00410 DIM PATT(3) : MAT READ PATT       ! Dimension, read array
00420 DATA 16,16,16
00430 CALL GDDM ('CHPAT',3,PATT())       ! Set all patterns to solid
00440 ! ***************** Legend *******************************
00450 INTEGER KATT                      ! Declare key attribute array
00460 DIM KATT(2) : MAT READ KATT       ! Dimension, read array
00470 DATA 6,2
00480 ! 6 = yellow, 2 = character mode
00490 CALL GDDM ('CHKATT',2,KATT())      ! Set key label attributes
00500 CALL GDDM ('CHSET','KBOX')         ! Enclose legend in box
00510 CALL GDDM ('CHKEY',3,4,'S/3830813033') ! Three legend keys
00520 ! ********* Number of components ***************************
00530 CALL GDDM ('CHNUM',3)              ! Use three bars per x-value
00540 ! *********** Specify S/38 data ****************************
00550 DIM NMD(10) : MAT READ NMD        ! Dimension and read data array
00560 DATA  15,56,67,76,43,45,56,87,56,34
00570 ! ************* Draw chart *********************************
00580 CALL GDDM ('CHBAR',1,10,NMD())
00590 ! Draw chart with 10 data groups of 1 bar each
00600 ! *********** Specify 3081, 3033 data **********************
```

```
00610 DIM NAD(20) : MAT READ NAD        ! Dimension and read data array
00620 DATA  25,67,69,65,55,67,69,72,56,34
00630 DATA  35,62,73,64,43,65,66,68,66,45
00640 CALL GDDM ('CHBAR',2,10,NAD())
00650 ! Draw chart with 10 data groups of 2 bars each
00660 ! *************** Send to display ************************
00670 INTEGER ATTYPE,ATMOD,COUNT
00680 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT) ! Send to display
00690 ! ***************** TERMINATE ****************************
00700 CALL GDDM ('FSTERM')              ! Terminate graphics
00710 END                               ! End BASIC program
```

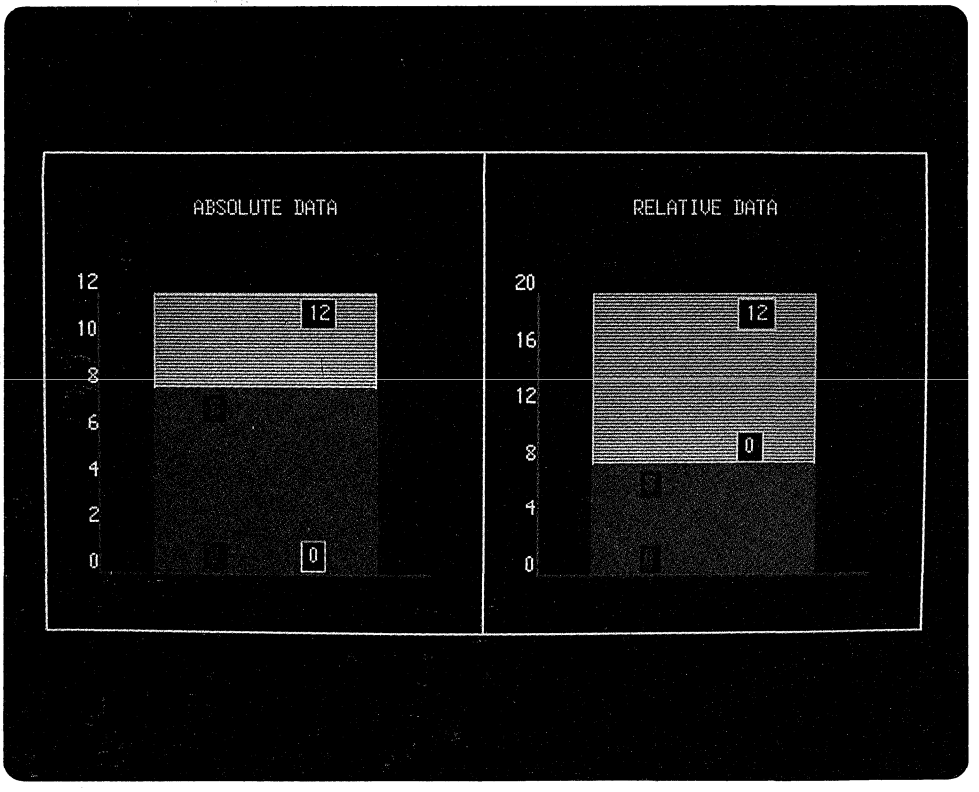
Drawing Composite-Bar Charts

Data for composite or floating-bar charts can be shown as *absolute* or *relative*.

Charts that use absolute data (the default) have the upper boundary of the shaded region showing the true values of the y-data for that component. All the data groups are plotted from the x axis.

Charts that use relative data have the upper boundary of a component showing the sum of its own y-values added to those of the components shown below it. The true y-value of the component can be seen by comparing the thickness of its shaded region with the y-axis scale. Each data group is plotted from the top of the previous one.

If each data group is larger than the one before, use absolute data. If some data groups are smaller than the ones before, they will be hidden if you use absolute data, use relative instead. For example:

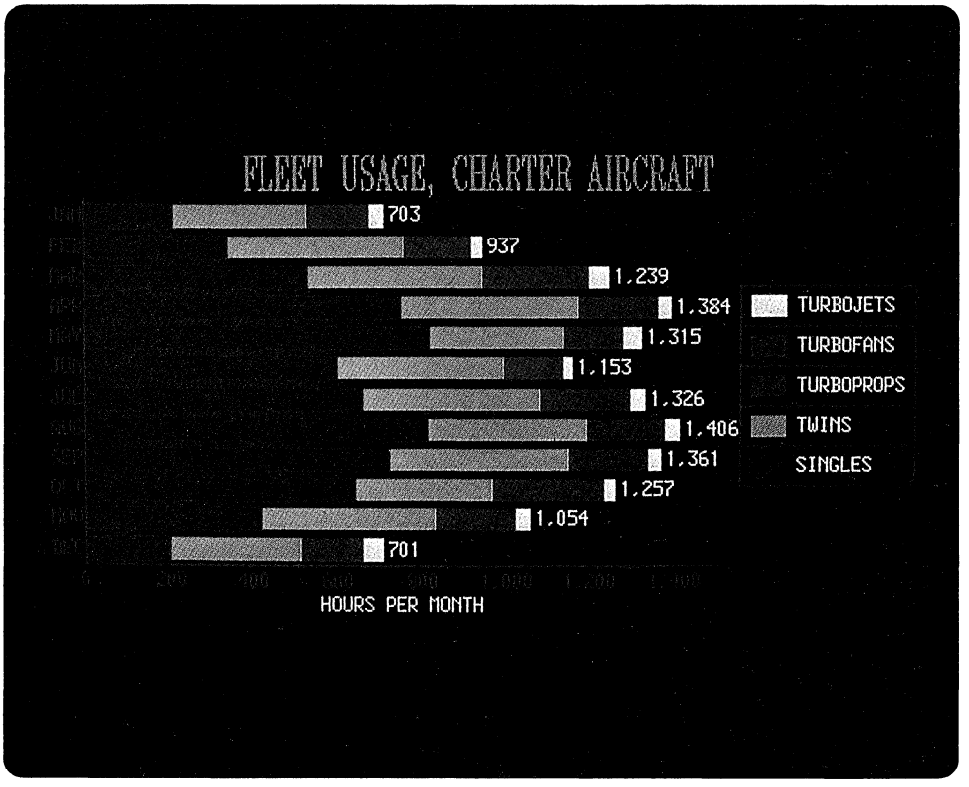


CHSET – Specify chart options. CHSET (CBAR) draws a composite-bar chart when CHBAR is called.

CHSET – Specify chart options. CHSET (INFILL) suppresses the shading of the first component.

CHSET – Specify chart options. CHSET (RELATIVE) shows the data as relative data.

This is an example of a horizontal composite-bar chart. The axis tick marks have been suppressed, and the color selection order has been modified:



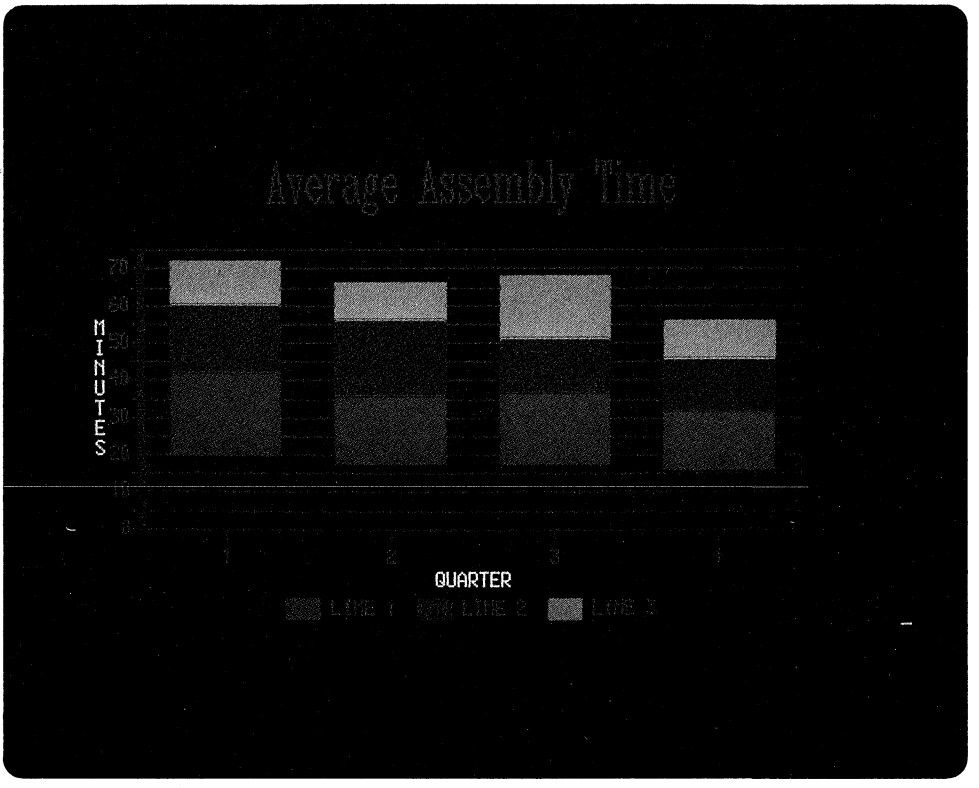

```
00010 ! ************* FLEET - COMPOSITE-BAR *********************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ***************** Symbol set ****************************
00060 CALL GDDM ('GSLSS',2,'ADMUWTRP',66)
00070 ! Load vector symbol set ADMUWTRP as symbol set #66
00080 ! **********  DEFINE THE CHART LAYOUT *********************
00090 CALL GDDM ('CHHMAR',3,0)          ! Change horizontal margins
00100 CALL GDDM ('CHVMAR',4,18)         ! Change vertical margin space
00110 INTEGER HATT                      ! Declare heading attribute
00120 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00130 DATA 2,3,66,250
00140 ! 2 = red, 3 = char mode, 66 = symbol set number, 250 = size
00150 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00160 CALL GDDM ('CHHEAD',29,'FLEET USAGE, CHARTER AIRCRAFT')
00170 ! Write 29-character heading with 'character string'
00180 CALL GDDM ('CHSET','XVERTICAL')   ! Horizontal orientation
00190 CALL GDDM ('CHYRNG',0.0,1500.0)   ! Set y-axis scale range
```

```
00200 ! ******************** Axes ******************************
00210 INTEGER TATT                        ! Declare axis title attribute
00220 DIM TATT(2) : MAT READ TATT         ! Dimension, read array
00230 DATA 6,2
00240 ! 6 = yellow, 2 = char mode
00250 CALL GDDM ('CHTATT',2,TATT())       ! Set title attributes
00260 CALL GDDM ('CHYTTL',15,'HOURS PER MONTH')  ! y-axis title
00270 INTEGER LATT                        ! Declare axis label attribute
00280 DIM LATT(4) : MAT READ LATT         ! Dimension, read array
00290 DATA 4,2,0,200
00300 ! 4 = green, 2 = character mode, 0 = symbol set, 200 = size
00310 CALL GDDM ('CHLATT',2,LATT())       ! Set label attributes
00320 CALL GDDM ('CHXMTH',1)              ! Use month labels
00330 CALL GDDM ('CHXSET','PLAIN')        ! Suppress x-axis tick marks
00340 CALL GDDM ('CHYSET','PLAIN')        ! Suppress y-axis tick marks
00350 ! ***************** Components ****************************
00360 INTEGER PATT                       ! Declare pattern array
00370 DIM PATT(5) : MAT READ PATT        ! Dimension, read array
00380 DATA 16,16,16,16,16
00390 CALL GDDM ('CHPAT',5,PATT())       ! Set all patterns to solid
00400 INTEGER COL                        ! Declare color array
00410 DIM COL(5) : MAT READ COL          ! Dimension, read array
00420 DATA 1,2,4,5,7
00430 CALL GDDM ('CHCOL',5,COL())        ! Set color selection order
00440 ! ****************** Bar values ***************************
00450 INTEGER VATT                        ! Declare value text attribute
00460 DIM VATT(2) : MAT READ VATT         ! Dimension, read array
00470 DATA 7,2
00480 ! 7 = white, 2 = character mode
00490 CALL GDDM ('CHVATT',2,VATT())       ! Set value text attributes
00500 CALL GDDM ('CHSET','VALUES')        ! Show bar values
00510 ! ***************** Legend ********************************
00520 INTEGER KATT                        ! Declare legend text array
00530 DIM KATT(2) : MAT READ KATT         ! Dimension, read array
00540 DATA 7,2
00550 ! 7 = white, 2 = character mode
00560 CALL GDDM ('CHKATT',2,KATT())       ! Set legend text attributes
00570 CALL GDDM ('CHSET','KBOX')          ! Enclose legend in box
00580 DIM CHAR$*50
00590 CHAR$ = 'SINGLES   TWINS     TURBOPROPSTURBOFANS TURBOJETS '
00600 CALL GDDM ('CHKEY',5,10,CHAR$)  ! Five 10-character legend keys
00610 ! *************** Specify data ****************************
00620 DIM HOURS(60) : MAT READ HOURS  ! Dimension and read data array
00630 DATA  210, 340, 530, 750, 818, 600, 659, 815, 723, 643, 420, 200
00640 DATA  315, 416, 410, 418, 318, 390, 419, 375, 422, 323, 410, 310
00650 DATA  100, 120, 200, 118,  98, 120, 149, 135, 122, 223, 135, 105
00660 DATA   45,  38,  56,  74,  43,  23,  65,  49,  72,  45,  55,  43
00670 DATA   33,  23,  43,  24,  38,  20,  34,  32,  22,  23,  34,  43
00680 !     JAN  FEB  MAR  APR  MAY  JUN  JUL  AUG  SEP  OCT  NOV  DEC
00690 ! ***************** Draw chart ****************************
00700 CALL GDDM ('CHSET','CBAR')        ! Draw chart as composite-bar
00710 CALL GDDM ('CHSET','RELATIVE')    ! Draw bars as relative data
00720 CALL GDDM ('CHBAR',5,12,HOURS())
00730 ! Draw chart with 12 bars made of 5 data groups each
00740 CALL GDDM ('CHDRAX')              ! Draw axes
00750 ! *************** Send to display *************************
00760 INTEGER ATTYPE,ATMOD,COUNT
00770 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)     ! Send to display
00790 CALL GDDM ('FSTERM')                        ! Terminate graphics
00800 END                                         ! End BASIC program
```

Drawing Floating-Bar Charts

CHSET — Specify chart options. CHSET (FBAR) draws a floating-bar chart when CHBAR is called.

Floating composite-bar charts can be used to show the individual ranges of value within a larger entity, in the same way as floating surface charts. The principles involved are the same, except that for the bar chart, you define the chart type with CHSET(FBAR). This an example of a floating-bar chart:




```
00010 ! *************** ASSEMBLY TIME ************************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics
00040 OPTION BASE 1                     ! Set array subscript base
00060 CALL GDDM ('GSLSS',2,'ADMUWCRP',66)
00070 ! Load vector symbol set ADMUWCRP as symbol set #66
00080 ! **********  DEFINE THE CHART LAYOUT *********************
00090 ! ***************** Heading *******************************
00100 INTEGER HATT                      ! Declare heading attribute
00110 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00120 DATA 5,3,66,300
00130 ! 5 = turq, 3 = char mode, 66 = symbol set number, 300 = size
00140 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00150 CALL GDDM ('CHHEAD',21,'Average Assembly Time')
00160 ! Write 21-character heading with 'character string'
00170 INTEGER PAT                       ! Declare pattern array
00180 DIM PAT(3) : MAT READ PAT         ! Dimension, read array
00190 DATA 16,16,16
00200 CALL GDDM ('CHPAT',3,PAT())       ! Set pattern attributes
00210 INTEGER COL : DIM COL(3) : MAT READ COL   ! Colors array
00220 DATA 2,3,1
00230 CALL GDDM ('CHCOL',3,COL())        ! Set color attributes
```

```
00240 ! ******************** Axes ******************************
00250 INTEGER AATT                        ! Declare axis attribute array
00260 DIM AATT(6) : MAT READ AATT         ! Dimension, read array
00270 DATA 1,0,0,1,0,0
00280 ! 1 = blue for both axes (0 = null)
00290 CALL GDDM ('CHAATT',6,AATT())       ! Set axis attributes
00300 INTEGER TATT                        ! Declare axis title attribute
00310 DIM TATT(2) : MAT READ TATT         ! Dimension, read array
00320 DATA 7,2
00330 ! 7 = white, 2 = character mode
00340 CALL GDDM ('CHTATT',2,TATT())       ! Set title attributes
00350 CALL GDDM ('CHXTTL',7,'QUARTER')    ! Write 7-character x-title
00360 CALL GDDM ('CHYTTL',7,'MINUTES')    ! Write 7-character y-title
00370 INTEGER LATT                        ! Declare label attribute
00380 DIM LATT(2) : MAT READ LATT         ! Dimension, read array
00390 DATA 5,2
00400 ! 5 = turquoise, 2 = character mode
00410 CALL GDDM ('CHLATT',2,LATT())       ! Set label attributes
00420 ! ***************** Legend ******************************
00430 INTEGER KATT                        ! Declare key label attribute
00440 DIM KATT(2) : MAT READ KATT         ! Dimension, read array
00450 DATA 5,2
00460 ! 5 = turquoise, 2 = character mode
00470 CALL GDDM ('CHKATT',2,KATT())       ! Set key label attributes
00480 CALL GDDM ('CHKEYP','H','B','C')
00490 ! Legend position Horizontal, Bottom, Centered
00500 CALL GDDM ('CHKEY',4,6,'      LINE 1LINE 2LINE 3')
00510 ! Write 4 6-character legend key labels using 'string' (1st null)
00520 ! ******************** Grid ******************************
00530 INTEGER GATT                        ! Declare axis attribute array
00540 DIM GATT(4) : GATT(4) = 1           ! y-axis grid is blue
00550 CALL GDDM ('CHGATT',4,GATT())       ! Set grid attributes
00560 CALL GDDM ('CHYTIC',5.0,4.0)        ! Add minor tick marks
00570 ! ********* Specify data for Surface Chart ***************
00580 DIM MINUTES(16) : MAT READ MINUTES     ! Array for Y axis
00590 DATA 20,18,18,16,22,18,19,16,18,20,14,14,12,10,17,10
00600 ! ************ Set attributes for Bar Chart ***************
00610 CALL GDDM ('CHSET','FBAR')             ! Draw floating-bar chart
00620 CALL GDDM ('CHYSET','GRID')            ! Draw grid for y axis
00630 CALL GDDM ('CHSET','RELATIVE')         ! Show relative data
00640 ! *************** DRAW SURFACE CHART ***********************
00650 CALL GDDM ('CHBAR',4,4,MINUTES())
00660 ! Draw chart with 4 bars, each with 4 parts (one invisible)
00670 ! *************** Send to display **************************
00680 INTEGER ATTYPE,ATMOD,COUNT                ! Declare integers
00690 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)    ! Send to display
00700 ! ***************** TERMINATE ******************************
00710 CALL GDDM ('FSTERM')                      ! Terminate graphics
00720 END                                       ! End BASIC program
```

Drawing Pie Charts

Setting the Component Color Selection Order

CHCOL – Set component color. CHCOL sets the color of the pie slices by defining a table that holds the number of colors and the order of their selection.

If CHCOL is not specified, the sequence of colors in the default color table is used. The default color table is, in this case, the GDDM color table which is either the default color table for the current page or a color table modified for use in the current page.

Setting the Shading Attributes

CHPAT – Set component shading pattern. CHPAT sets the type of pattern used by shaded components by defining a table that holds the number of patterns and the order of their selection.

If CHPAT is not specified, the sequence of patterns in the default pattern table is used. The default pattern table shown for the discussion of the GSPAT routine in Chapter 3, "Using GDDM" shows the order of selection.

CHSET – Specify chart options. CHSET (NOFILL) suppresses the shading of pies.

Writing Pie Chart Text

Besides using a chart heading and using chart notes to write text for pie charts, you can use a legend to identify the value and meaning of pie slices or you can write the legend key values next to the pie slices. For either of these options, you can also write the percentage values of each pie slice next to it, and connect the two with a line.

Using a legend

CHSET – Specify chart options. CHSET (PIEKEY) generates a chart legend, as opposed to writing the legend key labels next to the pie slices with CHSET (SPIDER). PIEKEY is the default.

When you use a legend, the attributes for the legend key labels are specified by the CHKATT routine.

Using legend key labels as pie slice labels

CHSET – Specify chart options. CHSET (SPIDER) writes the legend key labels next to the pie slices.

When you use a legend key labels as pie slice values, the attributes for the legend key labels are specified by the CHKATT routine.

Using value text as pie slice labels

You can specify that the value of each pie slice is written next to it. The value is connected to the slice with a line called a *spider tag*.

You can use value text in addition to the legend or the legend key labels. However, the routine used to set attributes for the value text differs with each.

For value text used when a *legend* is also used, use the CHVATT routine to set the color, character mode, symbol set used, and character size of the value text. For the color value to have an effect, SPILABEL must also be specified; otherwise, (when SPISECTOR is used) the value text is the color of the associated pie slice.

For value text used when *legend key labels* are used (SPIDER), use the CHKATT routine to set the color, character mode, symbol set used, and character size of the value text. For the color value to have an effect on both the value text and the legend key labels, SPILABEL must also be specified; otherwise, (when SPISECTOR is used) the value text and the legend key labels are the color of the associated pie slice.

Setting pie slice data values

CHSET – Specify chart options. CHSET (VALUES) writes the data value represented by each pie slice adjacent to it.

CHSET (BVALUES) blanks the areas where pie slice values are written. When the area is blanked, no other display feature can occupy the pie slice value text box. When the area is not blanked, the values can *overpaint* another feature.

CHVCHR – Number of value text characters. CHVCHR sets the number of characters to be used for showing pie slice values on a pie chart. The value text attributes are controlled by CHVATT. By default, up to 9 character positions can be used.

CHVATT – Value text attributes. CHVATT specifies the attributes for data value text. Data value text is used to show individual values for pie slices. Attributes that can be set are color, character mode, and character font and size. Character rotation is ignored for pie charts.

If CHVATT is not specified, each value uses standard-size device default characters of the default color, or of the color of the associated pie slice (if SPISECTOR is used).

CHSET – Specify chart options. CHSET (**SPISECTOR**|SPILABEL) specifies the color of spider tags and value text. If SPISECTOR is specified, the spider tags, value text, and legend key labels are colored the same as the associated sector. If SPILABEL is specified, the spider tags are the default color and the value text is either the color of the legend key labels (CHKATT) or the color specified by CHVATT for value text attributes.

Note: If spider tags overlap with the legend, you should move the legend (with CHKEYP) or change the size of the margins (with CHHMAR or CHVMAR).

Setting the Type of Data to be Shown

CHSET – Specify chart options. CHSET (**PERPIE**|ABPIE) shows the slices of the pie as relative data (PERPIE), or as absolute data (ABPIE). When the data is shown as relative data, each pie slice is shown with its true value (the value used in the pie chart data value array) as a percentage. Percentage values are subject to rounding errors and may not be precise. If the percentage values should be of decimal precision, you should calculate them in your program and then use them as pie slice labels. If the sum of all the values is less than 100, an incomplete pie is drawn. If the sum of the values exceeds 100, an error occurs and the pie is not drawn.

For absolute data (ABPIE), the data values of the pie chart array are reduced or increased so that their sum equals 100, which results in a complete pie.

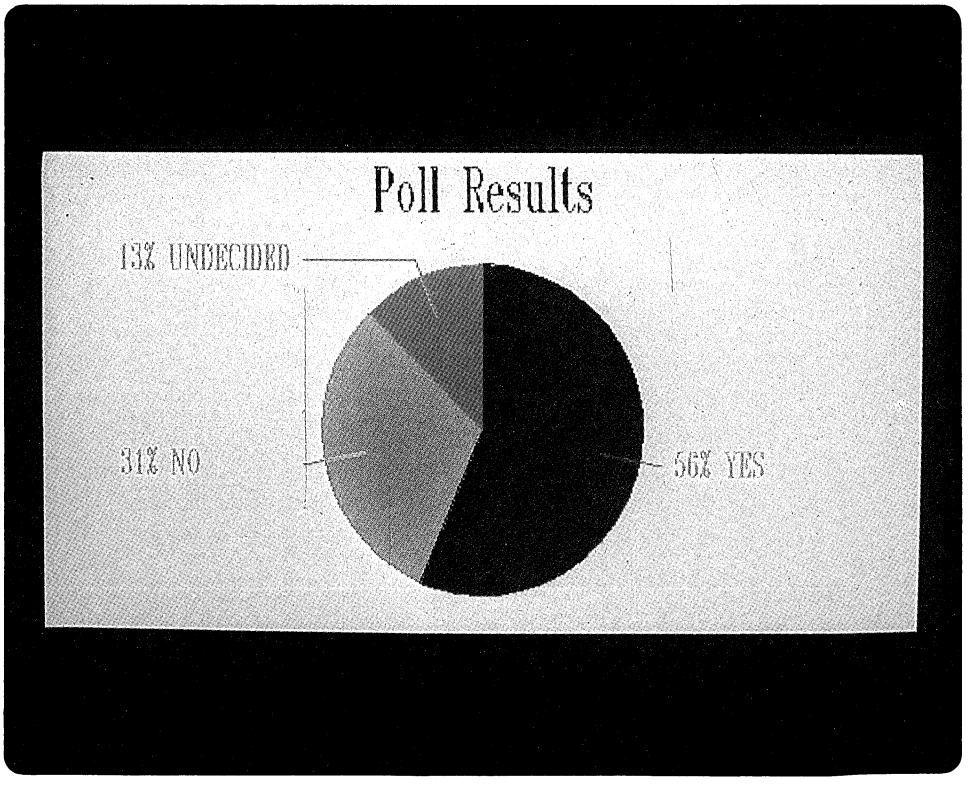
Controlling Pie Slices

CHPCTL – Specify chart appearance. You can make a pie chart more interesting by moving one or more of the slices out from the rest (exploding the chart). To specify which slices you want to move, use CHPEXP.

Drawing the Pie Chart

CHPIE – Draw a pie chart. CHPIE draws one or more pie charts. Parameters in the CHPIE routine specify the number of components (pies) to be drawn, the number of slices in each pie, and the array used for the dependent variables (the size of the pie slices).

The following chart is an example of a single-pie chart. The pie chart uses a background, and the patterns used for the pie slices have been set to solid. Also, both the horizontal and vertical margins have been reduced to zero to enlarge the pie:



```
00010 ! ***************** POLL RESULTS ***************************
00020 ! ***************** INITIALIZE *****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ***************** Symbol set *****************************
00060 CALL GDDM ('GSLSS',2,'ADMUWTRP',66)
00070 ! Load vector symbol set ADMUWTRP as symbol set #66
00080 ! **********  DEFINE THE CHART LAYOUT **********************
00090 INTEGER HATT                      ! Declare heading attribute
00100 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00110 DATA 1,3,66,325
00120 ! 1 = blue, 3 = char mode, 66 = symbol set number, 325 = size
00130 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00140 CALL GDDM ('CHHEAD',12,'Poll Results')
00150 ! Write 12-character heading with 'character string'
00160 CALL GDDM ('CHSET','CBACK')       ! Use background
00170 CALL GDDM ('CHHMAR',0,0)          ! Reduce margins
00180 CALL GDDM ('CHVMAR',0,0)          ! Reduce margins
00190 ! ***************** Pie labels *****************************
00200 INTEGER KATT                      ! Declare heading attribute
00210 DIM KATT(4) : MAT READ KATT       ! Dimension, read array
00220 DATA 2,3,66,200
00230 ! 2 = red, 3 = char mode, 66 = symbol set number, 200 = size
00240 CALL GDDM ('CHKATT',4,KATT())   ! Set key label attributes
00250 CALL GDDM ('CHSET','VALUES')    ! Show pie percentage values
00260 CALL GDDM ('CHSET','SPIDER')    ! Write legend key labels
00270 CALL GDDM ('CHSET','SPILABEL')  ! Legend key labels
00280 ! next to pie slices instead of as legend
00290 CALL GDDM ('CHKEY',3,9,'YES      NO       UNDECIDED')!Legend keys
00300 ! ***************** Specify data ***************************
00310 DIM POLL(3) : MAT READ POLL     ! Dimension and read data array
00320 DATA 45,25,10
00330 ! ***************** Draw chart *****************************
00340 INTEGER PAT                     ! Declare pattern attribute
00350 DIM PAT(3) : MAT READ PAT       ! Dimension, read array
00360 DATA 16,16,16
00370 ! Use solid fill for each pie slice
00380 CALL GDDM ('CHPAT',3,PAT())     ! Set heading attributes
00390 CALL GDDM ('CHSET','ABPIE')     ! Draw pie as absolute data
00400 CALL GDDM ('CHPIE',1,3,POLL())  ! Draw 1 pie, 3 slices
00410 ! *************** Send to display ************************
00420 INTEGER ATTYPE,ATMOD,COUNT                ! Declare integers
00430 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)   ! Send to display
00440 ! ***************** TERMINATE ******************************
00450 CALL GDDM ('FSTERM')                      ! Terminate graphics
00460 END                                       ! End BASIC program
```

Drawing a Multiple-Pie Chart

For other chart types, such as line charts and bar charts, the term *chart component* refers to a representation of one data group for the chart, such as a line or a bar. For pie charts, a chart component is one pie because one CHPIE routine can draw many pies.

There are two types of multiple-pie chart:

- **Charts where each pie shows a variation of an amount**. An example of such a multiple-pie chart is one that uses three pies to show the percentages of the total operating costs for a business over three years, see the example of a multiple-pie chart on page 4-80.

  For this type of chart, use a single CHPIE routine, but specify the number of components (pies) to be drawn. When drawing multiple-pie charts this way, you can specify CHSET(PROPIE) to vary the size of each chart based on the sum of its chart data values to show the value of each pie relative to the others.

  CHNUM and a corresponding number of CHPIE routines can also be used.

- **Charts where each pie shows a different group of data**. For an example of such a chart, see the example of a multiple-pie chart on page 4-82.

  For this type of chart, use CHNUM to specify the number of pies, then use a CHPIE routine to draw each pie. The relative size of the pies cannot be varied with CHSET(PROPIE).

Use these routines to draw multiple-pie charts:

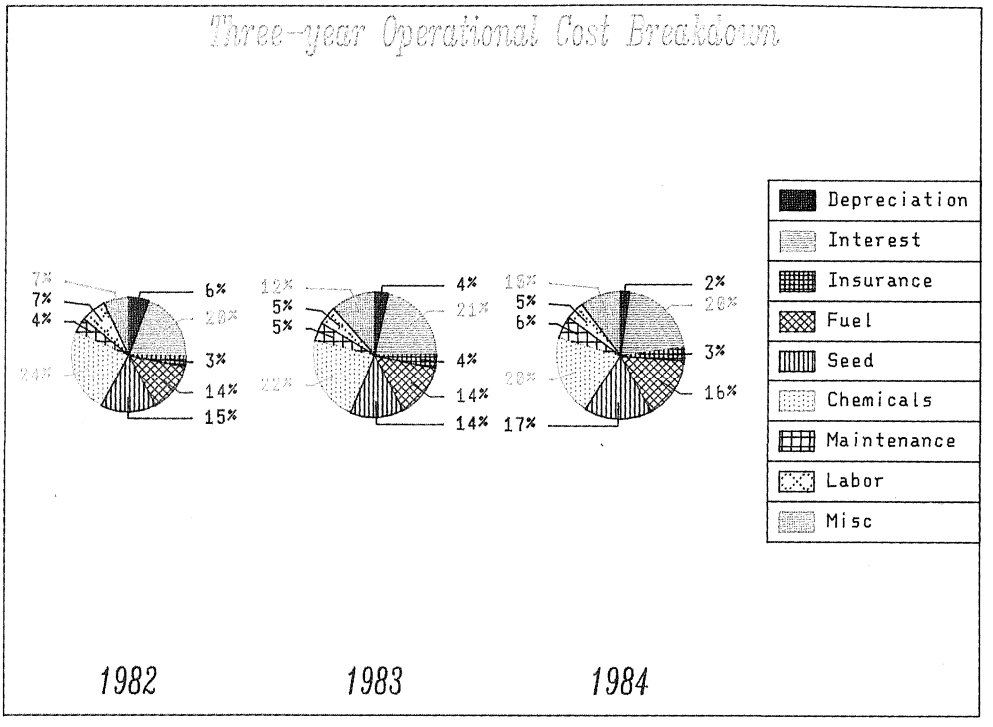
CHNUM – Set number of chart components. CHNUM sets the number of pie charts to be constructed on a single chart format. One pie chart is constructed by each call of the CHPIE routine.

CHPIER – Pie chart size reduction. The CHPIER routine specifies a reduction percentage for reducing the size of each chart.

CHSET – Specify chart options. CHSET(**ABPIE**|PROPIE) specifies whether the sum of the chart data determines the relative size (diameter) of each pie. ABPIE makes all pies the same size (the default), and PROPIE varies the size of the pies (the smallest-value pie will have the smallest diameter). PROPIE is valid only when a single CHPIE routine is used to draw multiple pies.

CHXLAB – x-axis label text specification. CHXLAB is used to specify the titles for the individual pies in multiple-pie charts.

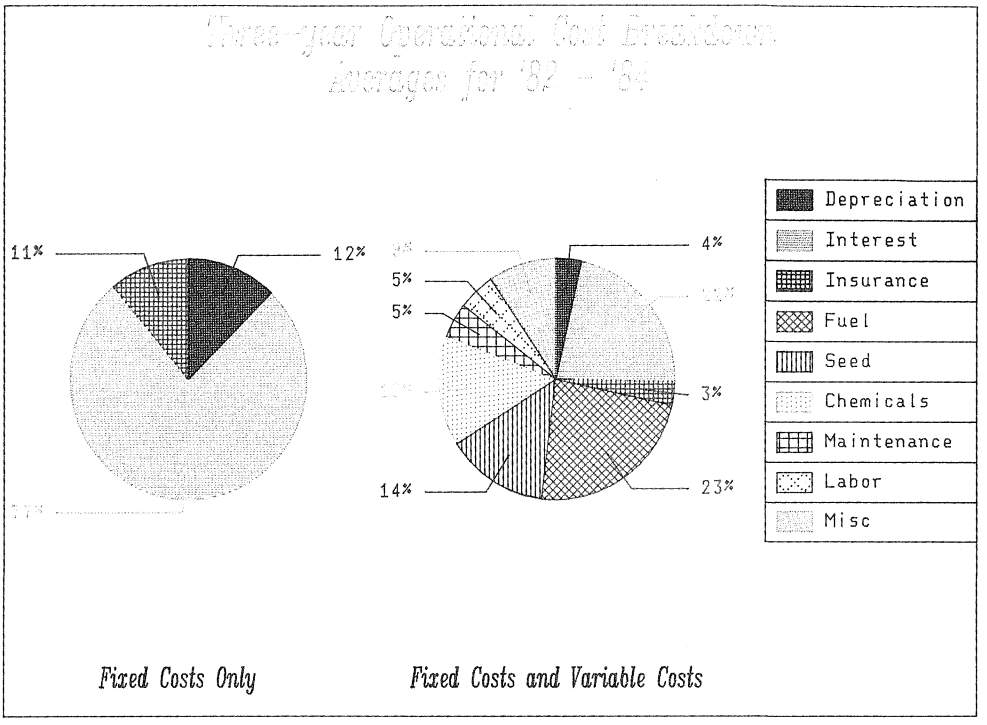
The following multiple-pie chart uses a single CHPIE routine to draw three pies that show variation of the same values over three years. The chart uses proportional sizes for the pies; the first pie is smaller than the second and third, because the sum of the data used for the pie is smaller than the data sums of the others. The vertical margins have been changed to allow room for the legend, which is drawn in reverse order:




```
00010 ! ***************** COST ANALYSIS *************************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 INTEGER PLST                      ! Declare integer
00060 DIM PLST(4) : MAT READ PLST       ! Dimension, read array
00070 DATA 11,50,16,1
00080 DIM NLST$(1) : NLST$(1) = ' '     ! Dimension, assign value
00090 CALL GDDM ('DSOPEN',2,1,'6180    ',4,PLST(),0,NLST$(1))
00100 ! Open plotter device 2 of family 1 named 6180,
00110 ! using PLST option group 11 value 50 (pen speed 50% of max),
00120 ! and using group 16 option 1 (horizontal paper orientation);
00130 ! name list has 0 names in array NLST$
00140 CALL GDDM ('DSUSE',1,2)
00150 ! Use device 2 as active device (option 1)
00160 ! ***************** Symbol set ****************************
00170 CALL GDDM ('GSLSS',2,'ADMUWCIP',66)
00180 ! Load vector symbol set ADMUWCIP as symbol set #66
00190 ! **********  DEFINE THE CHART LAYOUT *********************
00200 CALL GDDM ('CHSET','CBOX')      ! Use chart frame
00210 CALL GDDM ('CHSET','ABPIE')     ! Show absolute values
00220 CALL GDDM ('CHSET','PROPIE')    ! Pie size relative to sum of sl&
&ices
00230 CALL GDDM ('CHSET','VALUES')    ! Show pie values
00240 CALL GDDM ('CHHMAR',3,1)        ! Bottom margin 3, top margin 1
00250 CALL GDDM ('CHVMAR',0,20)       ! Left margin 0, right margin 20
```

```
00260 INTEGER HATT                    ! Declare heading attribute
00270 DIM HATT(4) : MAT READ HATT     ! Dimension, read array
00280 DATA 2,3,66,225
00290 ! 2 = pen 2, 3 = char mode, 66 = symbol set number, 225 = size
00300 CALL GDDM ('CHHATT',4,HATT())   ! Set heading attributes
00310 CALL GDDM ('CHHEAD',37,'Three-year Operational Cost Breakdown')
00320 ! Write 37-character heading with 'character string'
00330 ! ***************** Pie titles ****************************
00340 INTEGER LATT                    ! Declare label attribute
00350 DIM LATT(4) : MAT READ LATT     ! Dimension, read array
00360 DATA 5,3,66,200
00370 ! 5 = pen 5, 3 = character mode, 66 = symbol set, 200 = size
00380 CALL GDDM ('CHLATT',4,LATT())   ! Set label attributes
00390 CALL GDDM ('CHXLAB',3,4,'198219831984')
00400 ! Write three 4-character labels to be used as pie titles
00410 ! ***************** Legend ********************************
00420 CALL GDDM ('CHSET','KBOX')      ! Enclose legend in box
00430 DIM CHAR$*110
00440 CALL GDDM ('CHSET','KREVERSE')  ! Write legend reverse order
00450 CHAR$ = 'DepreciationInterest    Insurance   Fuel        Seed   &
&     Chemicals   Maintenance Labor       Misc        '
00460 CALL GDDM ('CHKEY',9,12,CHAR$)  ! Write legend keys
00470 ! ************** Specify chart data ************************
00480 DIM FARM(27) : MAT READ FARM    ! Dimension and read data array
00490 DATA 4,14,2,10,11,17,3,5,5
00500 DATA 3,18,3,12,12,19,4,4,10
00510 DATA 2,18,3,14,15,18,5,4,9
00520 CALL GDDM ('CHPIE',3,9,FARM())  ! Draw 3 pies
00530 ! ***************** Send to plotter ***********************
00540 CALL GDDM ('FSFRCE')            ! Send to plotter
00550 ! ****************** TERMINATE ****************************
00560 CALL GDDM ('FSTERM')            ! Terminate graphics
00570 END                             ! End BASIC program
```

This multiple-pie chart uses CHNUM to specify two pies, and uses two corresponding CHPIE routines:




```
00010 ! *********** FIXED VERSUS VARIABLE ***********************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 INTEGER PLST                      ! Declare integer
00060 DIM PLST(4) : MAT READ PLST       ! Dimension, read array
00070 DATA 11,50,16,1
00080 DIM NLST$(1) : NLST$(1) = ' '     ! Dimension, assign value
00090 CALL GDDM ('DSOPEN',2,1,'6180    ',4,PLST(),0,NLST$(1))
00100 ! Open plotter device 2 of family 1 named 6180,
00110 ! using PLST option group 11 value 50 (pen speed 50% of max),
00120 ! and using group 16 option 1 (horizontal paper orientation);
00130 ! name list has 0 names in array NLST$
00140 CALL GDDM ('DSUSE',1,2)
00150 ! Use device 2 as active device (option 1)
00160 ! ***************** Symbol set ****************************
00170 CALL GDDM ('GSLSS',2,'ADMUWCIP',66)
00180 ! Load vector symbol set ADMUWCIP as symbol set #66
00190 ! **********  DEFINE THE CHART LAYOUT **********************
00200 CALL GDDM ('CHSET','CBOX')       ! Use chart frame
00210 CALL GDDM ('CHSET','ABPIE')      ! Show absolute values
00220 CALL GDDM ('CHSET','VALUES')     ! Show pie values
00230 CALL GDDM ('CHHMAR',3,1)         ! Bottom margin 3, top margin 1
00240 CALL GDDM ('CHVMAR',0,20)        ! Left margin 0, right margin 20
00250 INTEGER HATT                     ! Declare heading attribute
00260 DIM HATT(4) : MAT READ HATT      ! Dimension, read array
00270 DATA 2,3,66,225
00280 ! 2 = pen 2, 3 = char mode, 66 = symbol set number, 225 = size
00290 CALL GDDM ('CHHATT',4,HATT())    ! Set heading attributes
00300 DIM CHAR$*110
```

```
00310 CHAR$ = 'Three-year Operational Cost Breakdown;Averages for ''82&
& - ''84'
00320 CALL GDDM ('CHHEAD',60,CHAR$)   ! Write heading
00330 CALL GDDM ('CHNUM',2)           ! Draw 2 pie charts
00340 ! *************** Pie titles ****************************
00350 INTEGER LATT                     ! Declare label attribute
00360 DIM LATT(4) : MAT READ LATT      ! Dimension, read array
00370 DATA 5,3,66,150
00380 ! 5 = pen 5, 3 = character mode, 66 = symbol set, 150 = size
00390 CALL GDDM ('CHLATT',4,LATT())    ! Set label attributes
00400 CHAR$ = '      Fixed Costs Only        Fixed Costs and Variable &
&Costs'
00410 CALL GDDM ('CHXLAB',2,30,CHAR$)
00420 ! Write two 30-character labels to be used as pie titles
00430 ! ******************* Legend *****************************
00440 CALL GDDM ('CHSET','KBOX')       ! Enclose legend in box
00450 CALL GDDM ('CHSET','KREVERSE')   ! Write legend reverse order
00460 INTEGER KATT                     ! Declare label attribute
00470 DIM KATT(2) : MAT READ KATT      ! Dimension, read array
00480 DATA 0,2
00490 ! 0 = pen 1, 2 = character mode
00500 CALL GDDM ('CHKATT',2,KATT())    ! Set legend text attributes
00510 CHAR$ = 'DepreciationInterest    Insurance   Fuel        Seed   &
&     Chemicals   Maintenance Labor       Misc        '
00520 CALL GDDM ('CHKEY',9,12,CHAR$)   ! Write labels
00530 ! ************ Draw fixed-cost chart **********************
00540 DIM FIX(3) : MAT READ FIX        ! Dimension and read data array
00550 DATA  9,56,8
00560 CALL GDDM ('CHPIE',1,3,FIX())    ! Draw 1 pie, 3 slices
00570 ! ************ Draw variable cost chart ******************
00580 DIM VAR(9) : MAT READ VAR        ! Dimension and read data array
00590 DATA 9,56,8,60,36,38,12,13,24
00600 CALL GDDM ('CHPIE',1,9,VAR())    ! Draw 1 pie, 9 slices
00610 ! *************** Send to plotter ************************
00620 CALL GDDM ('FSFRCE')             ! Send to plotter
00630 ! ***************** TERMINATE ****************************
00640 CALL GDDM ('FSTERM')             ! Terminate graphics
00650 END                              ! End BASIC program
```

Drawing Histograms

Setting the Color of the Shaded Area

CHCOL – Set component color. CHCOL sets the color of the shaded area. The histogram uses the first color in the CHCOL color-selection table.

If CHCOL is not specified, the sequence of colors in the default color table is used. The default color table is, in this case, the GDDM color table which is either the default color table for the current page or a color table modified for use in the current page.

Setting the Shading Attributes

CHSET – Specify chart options. CHSET (NOFILL) suppresses the shading of the area.

CHPAT – Set component shading pattern. CHPAT sets the pattern of the shaded area. The histogram uses the first pattern in the CHPAT pattern-selection table.

Setting the Type of Data to be Shown

CHSET – Specify chart options. CHSET (RELATIVE) shows the data as relative data.

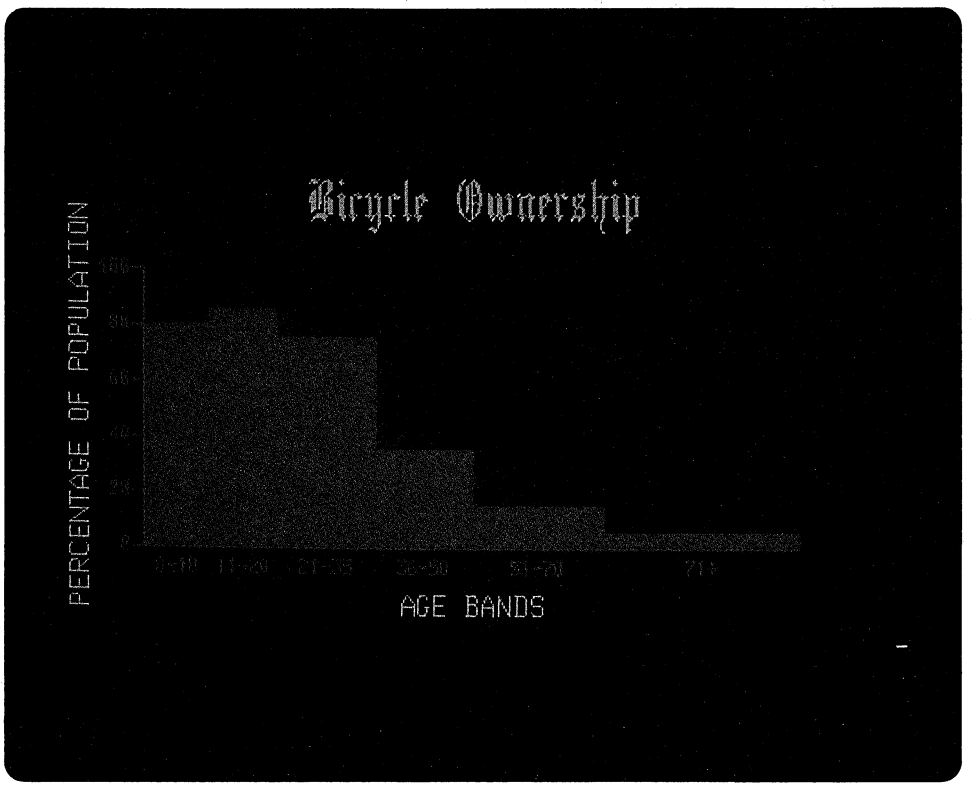
Suppressing the Risers

CHSET – Specify chart options. CHSET (**RISERS**|NORISERS) specifies whether lines (risers) are drawn between data groups or not. CHSET (NORISERS) stops the risers of the histogram being drawn. Histogram risers show the divisions between the steps of the histogram component levels.

Drawing the Histogram

CHHIST – Draw a histogram. CHHIST draws a histogram. Parameters in the CHHIST routine specify the number of components to be drawn, the number of ranges for each component, the arrays used for the ranges of the components, and the arrays used for the data groups the components are drawn to represent.

This is an example of a histogram. Chart notes are used in place of x-axis labels because the default labels are placed at even increments along the axis, while the components for the histogram use varying ranges along the x axis:




```
00010 ! ************** BICYCLE OWNERSHIP ************************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics.
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ***************** Symbol set ****************************
00060 CALL GDDM ('GSLSS',2,'ADMUWGEP',66)
00070 ! Load vector symbol set ADMUWGEP as symbol set #66
00080 ! **********  DEFINE THE CHART LAYOUT **********************
00090 INTEGER HATT                      ! Declare heading attribute
00100 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00110 DATA 2,3,66,300
00120 ! 2 = red, 3 = char mode, 66 = symbol set number, 300 = size
00130 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00140 CALL GDDM ('CHHEAD',17,'Bicycle Ownership')
00150 ! Write 17-character heading with 'character string'
00160 ! ****************** Axes *********************************
00170 INTEGER TATT                       ! Declare axis title attribute
00180 DIM TATT(4) : MAT READ TATT        ! Dimension, read array
00190 DATA 2,3,0,150
00200 ! 2 = red, 3 = character mode, 0 = symbol set, 150 = title size
00210 CALL GDDM ('CHTATT',4,TATT())      ! Set title attributes
00220 CALL GDDM ('CHYTTL',24,'PERCENTAGE OF POPULATION') ! y-axis title
00230 CALL GDDM ('CHXSET','NOLAB')       ! Suppress x-axis labels
00240 CALL GDDM ('CHXSET','PLAIN')       ! Suppress x-axis ticks
00250 ! ***************** Specify data **************************
00260 DIM XLO(6) : MAT READ XLO          ! Array for x low range
00270 DATA  0,10,20,35,50,70
```

```
00280 DIM XHI(6) : MAT READ XHI          ! Array for x high range
00290 DATA 10,20,35,50,70,100
00300 DIM YRNG(6) : MAT READ YRNG        ! Array for y range
00310 DATA 80,85,75,35,15,5
00320 ! ****************** Draw chart *************************
00330 CALL GDDM ('CHHIST',1,6,XLO(),XHI(),YRNG())
00340 ! Draw chart with 1 component of 6 ranges, using 3 arrays
00350 CALL GDDM ('CHDRAX')              ! Draw axes
00360 ! ****** Draw notes for labels and x-axis title ***********
00370 INTEGER NATT                      ! Declare note attributes
00380 DIM NATT(4) : MAT READ NATT       ! Dimension, read array
00390 DATA 4,2,0,0
00400 ! 4 = green, 2 = character mode, 0,0 = nulls
00410 CALL GDDM ('CHNATT',2,NATT())        ! Set note attributes
00420 CALL GDDM ('CHNOFF',5.01,4.5)        ! Set note position
00430 CALL GDDM ('CHNOTE','H2',4,'0-10')
00440 ! Write horizontal note, position H2, 4 characters of 'string'
00450 CALL GDDM ('CHNOFF',15.01,4.5)
00460 CALL GDDM ('CHNOTE','H2',5,'11-20')  ! Write note as x-label
00470 CALL GDDM ('CHNOFF',27.5,4.5)        ! Set note position
00480 CALL GDDM ('CHNOTE','H2',5,'21-35')  ! Write note as x-label
00490 CALL GDDM ('CHNOFF',42.5,4.5)        ! Position note
00500 CALL GDDM ('CHNOTE','H2',5,'36-50')  ! Write note as x-label
00510 CALL GDDM ('CHNOFF',60.01,4.5)       ! Position note
00520 CALL GDDM ('CHNOTE','H2',5,'51-70')  ! Write note as x-label
00530 CALL GDDM ('CHNOFF',85.01,4.5)       ! Position note
00540 CALL GDDM ('CHNOTE','H2',3,'71+')    ! Write note as x-label
00550 CALL GDDM ('CHNOFF',40.01,2.01)      ! Position note for title
00560 NATT(1)=2 : NATT(2)=3 : NATT(4)=150
00570 ! Change note attributes, 2 = red, 3 = mode 3, 150 = size
00580 CALL GDDM ('CHNATT',4,NATT())        ! Note will match y-title
00590 CALL GDDM ('CHNOTE','C5',9,'AGE BANDS')
00600 ! Write horizontal note, position C5, 9 characters of 'string'
00610 ! *************** Send to display ************************
00620 INTEGER ATTYPE,ATMOD,COUNT
00630 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)    ! Send to display
00640 ! ***************** TERMINATE *****************************
00650 CALL GDDM ('FSTERM')                       ! Terminate graphics
00660 END                                        ! End BASIC program
```

Drawing Venn Diagrams

Setting the Color of the Components

CHCOL – Set component color. CHCOL sets the color of the components. The diagram uses the first two colors in the CHCOL color-selection table.

If CHCOL is not specified, the sequence of colors in the default color table is used. The default color table is, in this case, the GDDM color table (the default color table for the current page or a color table modified for use in the current page).

Setting the Shading Attributes

CHSET – Specify chart options. CHSET (NOFILL) suppresses shading of the population circles and of the overlap area.

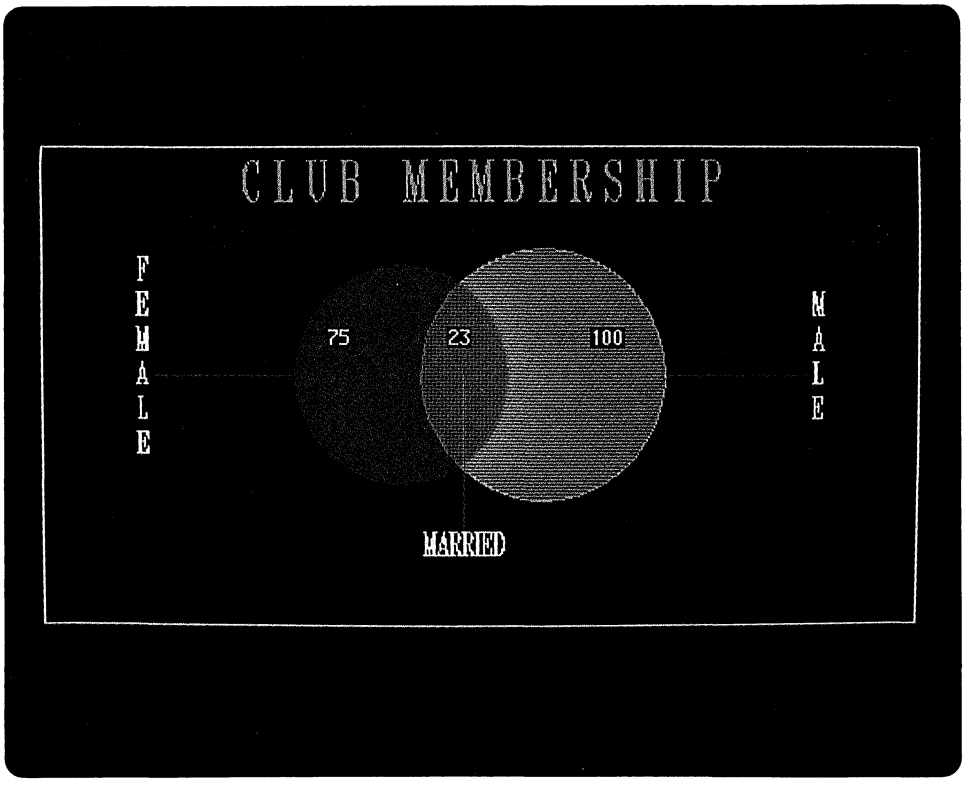
CHPAT – Set component shading pattern. CHPAT sets the pattern of the shaded area. The diagram uses the first two patterns in the CHPAT pattern-selection table.

If CHPAT is not specified, the sequence of patterns in the default pattern table is used. The default pattern table shown for the discussion of the GSPAT routine in Chapter 3, "Using GDDM" shows the order of selection.

Drawing the Venn Diagram

CHVENN – Draw a Venn diagram. CHVENN draws a Venn diagram. Parameters in the CHVENN routine specify the values of both populations, and the value of the overlap area.

This is an example of a Venn diagram. It has a chart frame, and uses blanked notes to indicate the relative values of the populations and their overlap:



```
00010 ! *************** CLUB MEMBERSHIP ************************
00020 ! ***************** INITIALIZE ****************************
00030 CALL GDDM ('FSINIT')              ! Initialize graphics
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ***************** Symbol set ****************************
00060 CALL GDDM ('GSLSS',2,'ADMUVTRP',66)
00070 ! Load vector symbol set ADMUVTRP as symbol set #66
00080 CALL GDDM ('GSLSS',2,'ADMUWCRP',67)
00090 ! Load vector symbol set ADMUWCRP as symbol set #67
00100 ! **********  DEFINE THE CHART LAYOUT *********************
00110 CALL GDDM ('CHSET','CBOX')        ! Enclose chart in frame
00120 INTEGER HATT                      ! Declare heading attribute
00130 DIM HATT(4) : MAT READ HATT       ! Dimension, read array
00140 DATA 2,3,66,300
00150 ! 2 = red, 3 = char mode, 66 = symbol set number, 300 = size
00160 CALL GDDM ('CHHATT',4,HATT())     ! Set heading attributes
00170 CALL GDDM ('CHHEAD',15,'CLUB MEMBERSHIP')
00180 ! Write 15-character heading with 'character string'
00190 ! ******************  Key  ********************************
00200 INTEGER KATT                       ! Declare key attribute array
00210 DIM KATT(4) : MAT READ KATT        ! Dimension, read array
00220 DATA 6,3,67,175
00230 ! 6 = yellow, 3 = char mode, 67 = symbol set number, 175 = size
00240 CALL GDDM ('CHKATT',4,KATT())      ! Set key attributes
00250 CALL GDDM ('CHKEY',3,7,'FEMALE  MALE  MARRIED')
00260 ! Write key, 3 elements, 7 characters of 'string'
00270 ! ************* Draw chart ********************************
00280 CALL GDDM ('CHVENN',75.0,100.0,23.0)  ! Draw chart using data
00290 ! ********** Draw notes to show populations ***************
00300 CALL GDDM ('CHSET','BNOTE')          ! Blank chart note areas
00310 INTEGER NATT                         ! Declare note attributes
00320 DIM NATT(2) : MAT READ NATT          ! Dimension, read array
00330 DATA 7,2
00340 ! 7 = white, 2 = character mode
00350 CALL GDDM ('CHNATT',2,NATT())        ! Set note attributes
00360 CALL GDDM ('CHNOFF',26.01,14.01)     ! Set note position
00370 CALL GDDM ('CHNOTE','C7',2,'75')     ! Write note for group 1
00380 ! Write horizontal note, position C7, 2 characters of 'string'
00390 CALL GDDM ('CHNOFF',37.0,14.01)      ! Set note position
00400 CALL GDDM ('CHNOTE','C7',2,'23')     ! Write note for overlap
00410 CALL GDDM ('CHNOFF',50.01,14.01)     ! Set note position
00420 CALL GDDM ('CHNOTE','C7',3,'100')    ! Write note for group 2
00430 ! *************** Send to display ************************
00440 INTEGER ATTYPE,ATMOD,COUNT
00450 CALL GDDM ('ASREAD',ATTYPE,ATMOD,COUNT)    ! Send to display
00460 ! ***************** TERMINATE *****************************
00470 CALL GDDM ('FSTERM')                       ! Terminate graphics
00480 END                                        ! End BASIC program
```

More Control Routines

"Control Operations" on page 4-13 discussed the control routines you use to initialize and otherwise control the characteristics of the program and picture *before* any chart definition or chart construction routines were called.

The following chart control routines, called *after* the chart-drawing routine has been specified, let you either draw another chart or exit the program.

After the chart has been defined and constructed, you can:

1. Reset the processing state so that other state-1 attributes and options can be set.

   When you reset the processing state, you can *modify* the chart or create a new chart of the same type quickly and efficiently within the same program.

2. Reinitialize Presentation Graphics and start over, without terminating the graphics environment.
3. Terminate the Presentation Graphics portion of the program, and then send the picture to a device (using the device control routines described in "Device Controls" on page 3-64).

The following routines can be used to perform these three actions:

Reset the Processing State

CHSTRT – Reset the processing state. If CHSTRT is specified *after* a chart-drawing routine, it resets the program to the point just prior to the most recent chart-drawing routine. All chart control and chart definition routines maintain their values. This routine is useful for a program that draws a number of similar charts.

If CHSTRT is specified *before* a chart-drawing routine, it reinitializes the program by resetting all chart definition routines to their default values.

Reinitialize Presentation Graphics

CHRNIT – Reinitialize Presentation Graphics. CHRNIT reinitializes the program by resetting all chart definition routines to their default values. CHRNIT can be used if another chart will be shown on the same screen, in the area specified by CHAREA.

Terminate Presentation Graphics

CHTERM – Terminate Presentation Graphics. CHTERM terminates the Presentation Graphics environment and releases storage used by Presentation Graphics to construct the chart. You should issue the CHTERM routine as soon as you know you are not going to call more Presentation Graphics routines. CHTERM can be issued before the GDDM routine ASREAD sends the chart to the device for display, or before other GDDM routines are used to define the picture in more detail.

Summary of This Chapter

In Chapter 2, "The Application Program Interface to Graphics" the program that drew the line chart introduced you to the basic idea of Presentation Graphics that routines are called from application programs written in high-level languages to draw a chart. When the chart is drawn, the program sends it to a display device.

Chapter 4, "Using Presentation Graphics" showed you the concepts and functions of Presentation Graphics routines. Specifically, the topics presented in this chapter were:

Understanding Presentation Graphics
The first part of the chapter showed you the various types of chart you can draw with Presentation Graphics routines. It also showed you some ways to use the charts.

Drawing charts with Presentation Graphics routines
The second part of the chapter showed the structure of Presentation Graphics programs, and the typical sequence of Presentation Graphics routines used in a program to draw a chart. Routines that define the chart were shown, as well as the routines that perform the chart construction. Explained last were other Presentation Graphics routines that control the environment.

At this point you should experiment with Presentation Graphics routines in simple programs to gain a better understanding of them, and to get ideas for application programs that use them. (Chapter 6, "Graphics Application Program Examples" offers more ideas for programs, and shows and explains some sample programs.)

Here are some points to remember:

1. FSINIT must be specified before any calls to GDDM or Presentation Graphics routines are performed by your program.
2. All of the chart definition routines can be allowed to default; however, the chart can be difficult to understand without chart text (axis titles, labels, and a chart legend).
3. Some chart definition routines can be used only in state-1 (before the chart drawing routine is used).
4. Data types must be declared according to the high-level language being used.

For more detailed information about each Presentation Graphics routine, refer to the *GDDM Programming Reference* manual.

Chapter 5. OS/400 Programming Considerations

In Chapter 3, "Using GDDM," and Chapter 4, "Using Presentation Graphics," you learned of the variety of GDDM and Presentation Graphics routines that you can use in application programs to construct graphics pictures and charts. The concepts and functions of GDDM and Presentation Graphics routines were discussed, but few ideas about how GDDM and Presentation Graphics routines work with the AS/400 System were discussed.

This chapter is about the AS/400 System and its interface to OS/400 Graphics. The topics are:

Files for graphics

Graphics symbol sets

Performance

Error recovery

User-defined data streams

AS/400 Files Used for Graphics

Three types of file can be used with graphics application programs:

1. Display device files, which can be used with graphics to provide more usable and attractive display formats for interactive applications.
2. Printer device files, which can be used to format output from a printer graphics application program for printing on a work station printer capable of graphics.
3. Database files, which can be used to hold the hexadecimal data of a graphics data format (GDF) file for later interpretation by the AS/400 System or another system.

Display Files

By using only GDDM and Presentation Graphics routines in your program, you can write programs that produce pictures and charts. If you want a program that combines OS/400 Graphics with work station input/output operations, you must use DDS (data description specification) display device files with your graphics program. (Ideas for programs that prompt for information, and then perform actions based on that information are given in Chapter 6, "Graphics Application Program Examples.")

For an OS/400 Graphics application program to perform input/output operations, the program must first open a display device file to communicate with a display device. The program can then do input and output operations (usually coded as READ and WRITE operations in high-level language programs) to transfer data to and from the device. For OS/400 Graphics, the preferred display device is the IBM PC with Workstation Function.

There are two types of display files that you can use for graphics applications on the AS/400 System:

IBM-supplied: QDGDDM is an IBM-supplied display file in library QSYS which is used when you call OS/400 Graphics routines for display or plotter devices. Usually, you are aware of this file only when other display files compete for the use of the same display device, or when its name appears in error messages.

© Copyright IBM Corp. 1991

Externally-described: You can describe your own display file with DDS and create it using the CRTDSPF (Create Display File) command. An externally-described display file that uses the ALWGPH DDS keyword allows you to display alphanumeric data and input fields at the same time graphics are being displayed. ALWGPH is described in "The ALWGPH DDS Keyword" on page 5-3.

When your graphics application program sends a picture to the graphics work station (with or without externally-described display files), or when a record is displayed that uses the ALWGPH keyword, the screen is set to *graphics display mode*.

In graphics display mode:

1. The device is automatically placed in *reduced line spacing* mode, with less space between lines. In graphics display mode, the reduced line spacing cannot be overridden by the work station user.
2. The *graphics display mode* indicator, the blue uppercase G character, is shown at the bottom of the screen.
3. The graphics picture is shown as a background to any alphanumeric data shown by an externally-described display file.

QDGDDM Display File Considerations

GDDM uses the QDGDDM display file for communication with display devices capable of showing graphics. These special considerations apply to the QDGDDM display file:

QDGDDM is secured from overrides; GDDM graphics output cannot be redirected to other devices by use of override commands, such as the OVRDSPF (Override with Display File) command. If GDDM output is desired on a specific device, you can specify a device identifier in the DSOPEN routine. The device must be made known to the system by using a CRTDEVDSP (Create Device Description (Display)) command.

If a file error occurs (perhaps due to a device failure), GDDM generates a severity 40 (unrecoverable) error. The application program must recover the device by doing a DSCLS-DSOPEN sequence to close and reopen the file.

If a process has performed a DSOPEN to the device, but not a DSCLS to release the device, an attempt by an alternative process (such as System Request) to use a graphics-capable device will fail.

QDGDDM competes with user-defined display files for graphics devices as follows:

When	This Happens with ASREAD	This Happens with FSFRCE
Display file is active, and allows graphics (ALWGPH has been specified). Graphics display mode is in effect.	1. Display file is suspended. 2. Alphanumeric characters are cleared. 3. Graphics are written. 4. Keyboard is unlocked. 5. Operator input is awaited. 6. Display file is restored. Graphics remain on screen, as background to the alphanumeric data displayed. Display file remains active.	1. Display file is suspended. 2. Graphics are written. 3. Display file is restored. Graphics remain on screen, as background to the alphanumeric data displayed. Alphanumeric characters are not cleared. Display file remains active.
Display file is active, and does not allow graphics (no ALWGPH keyword specified in display file). Graphics display mode is not in effect.	1. Display file is suspended. 2. Alphanumeric characters are cleared. 3. Graphics display mode is turned on. 4. Graphics are written. 5. Keyboard is unlocked. 6. Operator input is awaited. Only graphics remain on screen. Display file is not active.	1. Display file is suspended. 2. Alphanumeric characters are cleared. 3. Graphics display mode is turned on. 4. Graphics are written. 5. Keyboard is locked. Only graphics remain on screen. Display file is not active.
No display file is active. Graphics display mode is not in effect.	1. Screen is cleared. 2. Graphics display mode is turned on. 3. Graphics are written. 4. Keyboard is unlocked. 5. Operator input is awaited. Graphics remain on screen.	1. Screen is cleared. 2. Graphics display mode is turned on. 3. Graphics are written. 4. Keyboard is locked. Graphics remain on screen.

The ALWGPH DDS Keyword

With the ALWGPH keyword in your externally-described display files, you can display alphanumeric data at the same time you use display file QDGDDM to display graphics (your program uses internal display file QDGDDM).

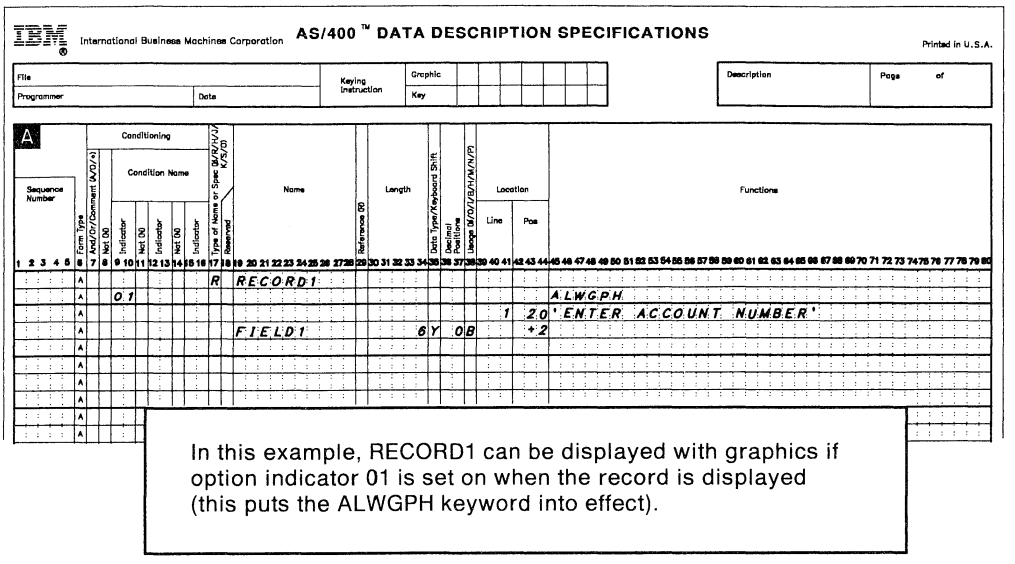
You must specify the DDS keyword ALWGPH in at least one record format in DDS for the externally-described file. Also, at least one record format that has the ALWGPH keyword must be displayed when graphics are shown.

When a record format that has the ALWGPH keyword is displayed on the graphics work station, the device is placed in graphics display mode (even if no other record format being shown has ALWGPH). If you use System Request (while the device is in graphics display mode) to start another job that uses other display files (with or without ALWGPH), those display files are also shown in graphics display mode.

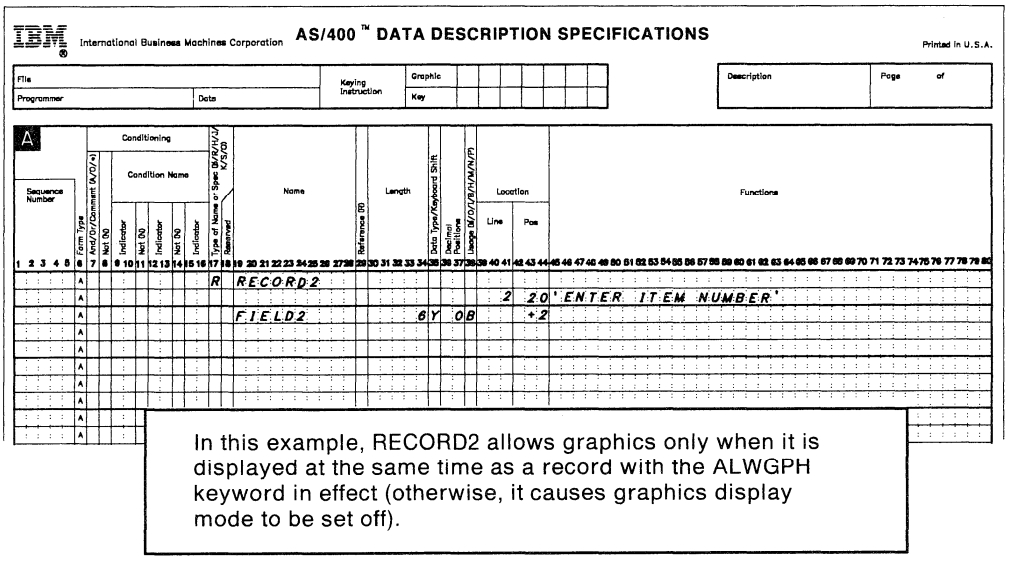
For plotters, the ALWGPH keyword has no effect and is ignored (alphanumeric data displayed on the graphics work station cannot be plotted).

For more information on specifying the ALWGPH keyword, refer to the *DDS Reference* manual.

The following example shows how to specify the ALWGPH keyword:



In this example, RECORD1 can be displayed with graphics if option indicator 01 is set on when the record is displayed (this puts the ALWGPH keyword into effect).



In this example, RECORD2 allows graphics only when it is displayed at the same time as a record with the ALWGPH keyword in effect (otherwise, it causes graphics display mode to be set off).

RV2S000-0

Printer Files

GDDM uses printer files to communicate with work station printers capable of graphics.

The device-token parameter of the DSOPEN routine identifies the program to GDDM as being one that is to produce printer output. When a DSOPEN routine has a printer device token, GDDM opens the default printer file QPGDDM. QSYS/QPGDDM is the default printer file, but you can create your own printer file (with characteristics similar to QPGDDM) and use it by passing its name to GDDM via the name-list parameter of the DSOPEN routine. For more information on using DSOPEN to open printer files, see Appendix A, "Devices Compatible with the AS/400 System."

QPGDDM Printer File Considerations

GDDM uses the QPGDDM printer file for communication with work station printers capable of graphics. When shipped, QPGDDM is defined as follows:

```
CRTPRTF FILE(QSYS/QPGDDM) +
        DEVTYPE(*SCS) +
        PAGESIZE(99 132) +
        LPI(9) +
        CPI(10) +
        OVRFLW(90) +
        RPLUNPRT(*NO) +
        CHRID(101 037) +
        SPOOL(*YES) +
        OUTQ(*JOB) +
        FILESEP(0) +
        SCHEDULE(*FILEEND) +
        LVLCHK(*NO) +
        SHARE(*NO) +
        TEXT('System printer file for printer graphics')
```

The size of the graphics page (where graphics are allowed to be displayed on the printer form) is determined by the PAGESIZE value for form width and the OVRFLW value in the printer file. QPGDDM specifies a PAGESIZE width of 132 characters (10 cpi) and an OVRFLW value of 90 lines (9 lpi). This means the default graphics page is 13.2 inches wide and 10 inches long.

The graphics page is positioned by its upper left corner being placed at the upper left corner of the printer form and its lower left corner being at the left edge of the OVRFLW line. If the user specifies a width and overflow covering the entire page, then the graphics page covers the entire form. The graphics field by default covers the entire graphics page.

The default coordinate range that is mapped on top of the graphics field is 0 through 100 in both the x and y directions. The origin (0,0) is in the lower left corner of the graphics field.

For example, if you specified a width of 130 characters (10 cpi) and an overflow value of 54 lines (9 lpi), then the graphics field would be 6 inches high by 13 inches wide, starting in the upper left corner of the printer form. If you did not call GSWIN in your application, then the coordinate range of this graphics field would be 0 through 100 in the x direction (horizontal) and 0 through 100 in the y direction (vertical), with the origin being in the lower left corner of the graphics field.

You can override any of the attributes of QPGDDM. The effect of overriding attributes for graphics using QPGDDM and user-defined print files is shown in this table:

PAGESIZE OVRFLW LPI CPI	When the PAGESIZE value for form width and the OVERFLW value are changed, graphics hierarchy default values (such as the default values for page size and graphic field size) are also changed. For example, a printer file that specifies a width of 50 columns and an OVRFLW of 25 lines causes a default graphics page of 25 rows by 50 columns. The default graphics field assumes the same dimensions, and the aspect ratio also changes. The lines-per-inch (LPI) value can be overridden, but has little effect other than to cause a different default page or field size. Any LPI value you use is converted to 9 LPI because graphics must be printed at 9 LPI. The default page or field size results from the conversion; if you use a printer file that has values of 6 LPI and OVRFLW 36, the page would eject after 6 inches of printing. When the conversion to 9 LPI is made, the OVRFLW value is converted to reflect the number of lines that can be printed on 6 inches (at 9 LPI, 54 lines can be printed on 6 inches). With a printer file in use or an override that uses the above values, a query of the default page or field will result in the converted values. The converted values are used only for the default page or field. If you define a page or field size less than the converted values, your values are used instead. You may want to specify your print file as 6 LPI instead of 9 LPI. For example, if you have forms which are 8.5 inches long, you cannot describe your form with 9 LPI (correct form size would be 76.5), but with 6 LPI you can by specifying a form size of 51 lines. The characters-per-inch (CPI) value for graphics printer files must be 10 or 15. The default character density is 10 CPI. For IPDS page printers, you probably want to change the PAGESIZE value for width and the OVRFLW value to match the size of the page you are printing on. For example, if you are using 8.5 inch by 11 inch paper, you may want to specify 83 for the width on the PAGESIZE value. This produces output that is 8.3 inches wide when the CPI value is 10. If the page is rotated 90 degrees, you may want to specify 110 for the width on the PAGESIZE value and 72 for the OVRFLW value. This produces output that is 11 inches wide (10 CPI) and 8 inches long (9 LPI).
PAGRTT	This parameter applies to the IPDS page printers. It specifies the degree of rotation of the output on the page with respect to the way the form is loaded into the printer. The default orientation of the page for IPDS page printers is vertical, as opposed to the landscape orientation of the plotters and SCS printers. A rotation value of 90 degrees produces a landscape orientation for your output, which is the same as for plotters and SCS printers.
SPOOL DEVTYPE OUTQ ALIGN	The SPOOL parameter works in conjunction with the DEVTYPE parameter. If you specify SPOOL(*NO) to send the file directly to the printer, DEVTYPE must specify the printer device name. SPOOL(*YES) causes the file to be spooled to the output queue named in the OUTQ parameter. ALIGN causes a forms alignment reply message to be send for SPOOL(*NO) files. Graphics printer files are generally incompatible with devices that do not match the device token used to produce the file. However, a printer file that is produced using the 522X device token can be printed on a 4234 printer, but not the converse. An SCS-type device token has precedence over the DEVTYPE (*IPDS) parameter. An IPDS device token has precedence over the DEVTYPE (*SCS) parameter.
MAXRCDS SCHEDULE COPIES FORMTYPE FILESEP HOLD SAVE	These parameters apply only if SPOOL(*YES) is specified. The parameters work in the same manner for graphics printer files as they do for other printer files.
PRTQLTY	This parameter applies to the 3812, 4214, and 4224 printers. It specifies the print quality for text on applications that merge text from non-graphics print files with graphics ones.

FORMFEED DRAWER	These parameters apply only to the IBM 4214 Printer, which can use cut sheets.
CHRID	This parameter applies only to the 3812 and 4224 printers. It specifies the character set to be used when using the default image symbol set.
SHARE	This parameter applies only to application programs that merge text from non-graphics print files with graphics printer files (SHARE(*YES)), and the open for the non-graphics print file must occur after the graphics printer file has been opened via the DSOPEN routine. For more detailed information, see "Merging Text and Graphics for Print Files" on page A-8.

Database Files

Database files on the AS/400 System can be used either to store data used as input by the graphics application program or to hold the graphics data format (GDF) file that is created by the program.

For examples of programs that receive input from database files, see Chapter 6, "Graphics Application Program Examples." For more information on GDF files, see "Using Graphics Data Format Files" on page 3-67.

OS/400 Graphics Symbol Sets

OS/400 Graphics uses *graphics symbol sets* (GSS) for text in GDDM and Presentation Graphics application programs. Each character in a graphics symbol set is a formatted group of EBCDIC code points.

Two types of graphics symbol sets are available:

Image symbols (mode-2 characters), where each character is built with a set of PELs

Vector symbols (mode-3 characters), where each character is built with a set of straight and curved lines

Image symbols are the default for GDDM, and vector symbols are the default for Presentation Graphics. For GDDM, however, you must explicitly set the character mode to 2 (with GSCM) before any of the attribute routines can have an effect.

Multinational character sets (the default for both image and vector symbols) containing the IBM multinational characters are shown in the *Programming Reference Summary* manual.

Characters from the multinational sets can be coded in hexadecimal from a graphics work station keyboard by first pressing the CMD key, then the accent key between the CMD key and the 1 key on the top row, and then entering the hexadecimal code. For example, the hexadecimal code for "A" is X'C1'.

Note: The characters in the multinational set might differ from the characters used in your country; if you need specific characters, use the appropriate national language symbol set for your country.

For IPDS Printers, the character set used by the default image symbol set (mode-2 characters) is determined by the CHRID parameter at the print file being used. For more information, see the CHRID parameter on the CRTPRTF command in the *CL Reference* manual.

For IBM-supplied image and vector symbol sets (found in library QGDDM), the prefix characters for symbol sets are as follows:

ADMM___	Multinational standard characters (the default vector symbols, and the default for Presentation Graphics on the graphics work station and for both GDDM and Presentation Graphics on the plotters)
ADMD___	Standard characters for each national language (vector symbols)

The following naming convention applies to vector symbol sets that have the specific font styles of Standard Simple, Standard Bold, Open Block, Filled Block, and Roman Principal:

ADMVM___	Multinational standard characters of a specific font style, mono-spaced
ADMWM___	Multinational standard characters of a specific font style, proportionally-spaced

National Language versions of these symbol sets are available. For more information, contact your IBM representative.

Vector symbol sets contain the following characters:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789
abcdefghijklmnopqrstuvwxyz $&*()-+=!:;',.?/
```

The prefix characters for vector symbol sets are:

ADMU___	Specialty characters
ADMUV___	Specialty characters, mono-spaced
ADMUW___	Specialty characters, proportionally-spaced

Most vector symbol sets come in two versions: mono-spaced and proportionally-spaced.

Mono-spaced characters in a character string use the same-sized character box (an "I" is as wide as an "M"), while proportionally-spaced characters take only as much space as the character itself.

More information on the uses and characteristics of both types of graphics symbols is available in "How to Draw Graphics Symbols" on page 3-25.

Image Symbol Sets

This table shows the image symbol sets (ISS, or mode-2 characters) available. They are all multinational standard characters and defaults for GDDM.

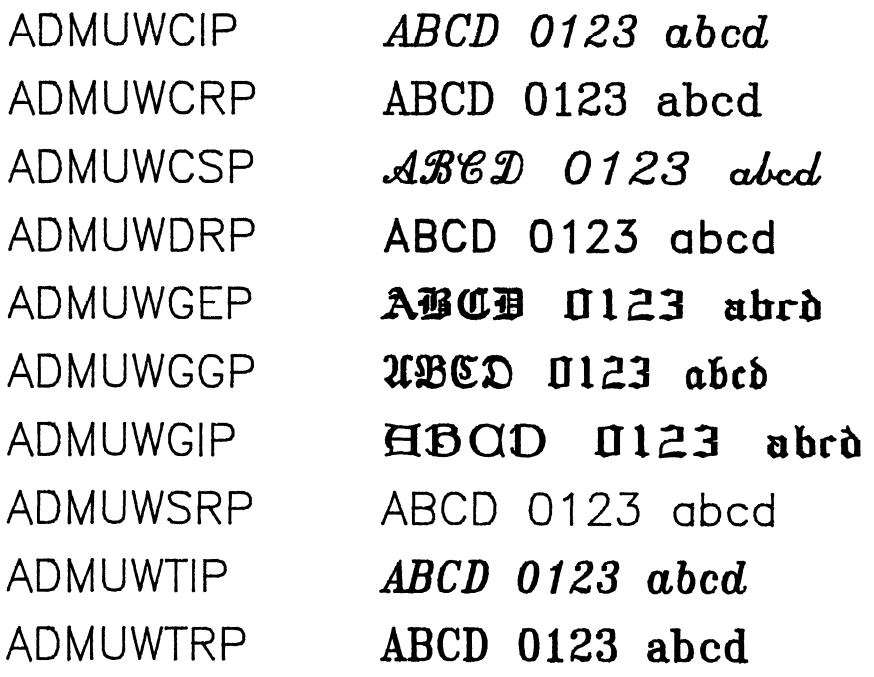
ISS name	Object text (description)
ADMMISSG	For the 4214 Printer
ADMMISSI	For the 5224/5 Printers, and the 4234 Printer at 10 cpi
ADMMISSP	For the 4234 Printer at 15 cpi

Note: The IPDS printers do not use a default image symbol set. When mode-2 characters are specified or defaulted, the printer hardware characters are used. The printer hardware character set used is determined by the CHRID parameter of the print file being used.

Vector Symbol Sets

Most vector symbol sets (VSS, or mode-3 characters) are available in *proportional* and *nonproportional* fonts. Proportional fonts are indicated by a W as the fifth character of the symbol set name, and nonproportional fonts are indicated by a V in that position.

Here are examples of the available symbol sets shown in their proportional form:



There is a set of standard multinational characters (ADMMVSS) that is available only in nonproportional font. Other multinational symbol sets are available as both proportional (W) and nonproportional (V) fonts:

ADMVMSS Multinational Standard Simple
ADMVMSB Multinational Standard Bold
ADMVMOB Multinational Open Block
ADMVMFB Multinational Filled Block
ADMVMRP Multinational Roman Principal

Finally, there are national language nonproportional symbol sets:

ADMDVSS American English standard characters
ADMDVSSE U.K. English standard characters
ADMDVSSF French standard characters
ADMDVSSG German standard characters
ADMDVSSI Italian standard characters
ADMDVSSK Katakana standard characters
ADMDVSSS Spanish standard characters

Using Graphics Symbol Sets

Graphics symbol sets are used to draw the character strings specified on the GSCHAR and GSCHAP GDDM routines, as well as for all of the text output produced by Presentation Graphics.

To use a specific mode-3 symbol set for GDDM, you must load the symbol set with the GSLSS routine and then select it with the GSCS routine. Also, you must set the character mode using GSCM.

Before you set attributes (except color) for mode-2 symbol set characters for GDDM, you must explicitly set the character mode with GSCM.

For Presentation Graphics, symbol sets loaded by GSLSS must be selected and attributes set by the appropriate CH__ATT routine (CHHAAT or CHLATT, for example).

Graphics symbol sets can also be used to draw the markers specified on the GSMARK and GSMRKS GDDM routines.

To use an image symbol set or a vector symbol set as a user-defined marker set, you must first load it as a marker set using the GSLSS routine. As soon as the marker symbol set is loaded, it is automatically selected for use as the current marker set.

Creating Graphics Symbol Sets

Symbol sets produced on the System/370 computer by the licensed programs *GDDM Image Symbol Editor* (mode-2 image characters) and *GDDM-PGF Vector Symbol Editor* (mode-3 vector characters) are fully compatible with the AS/400 System, as are symbol sets supplied with System/370 GDDM.

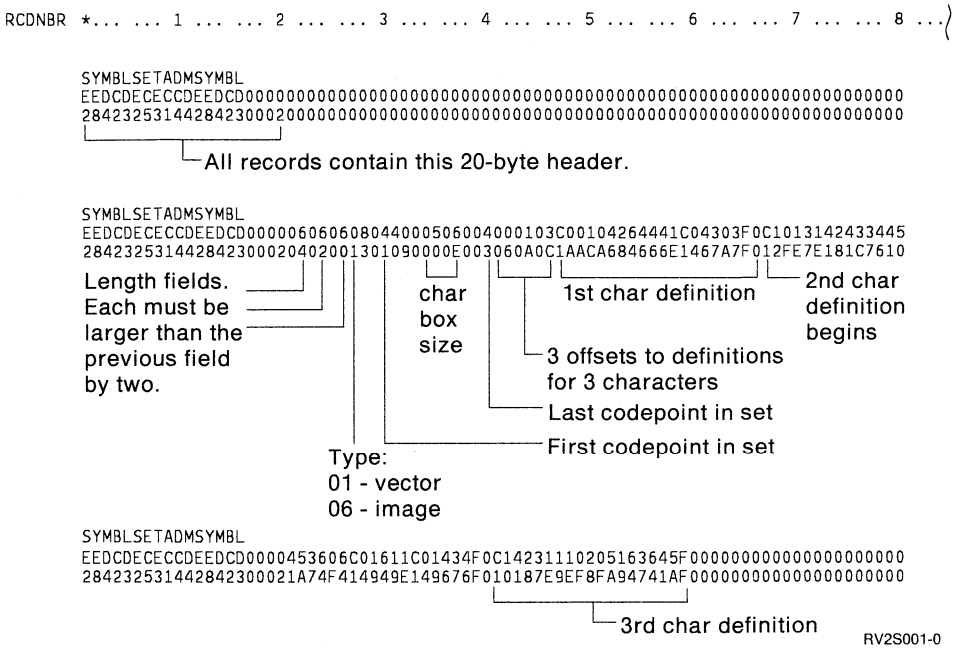
To use System/370 symbol sets in OS/400 Graphics, you must transport the source data from the System/370 computer to the AS/400 System, and then run the CRTGSS (Create Graphics Symbol Set) CL command to convert the symbol set source data into an AS/400 *GSS object type.

Because System/370 symbol set source data usually has a record length of 400, some communications links between the AS/400 System and the System/370 computer require that the source data be reformatted into 80-byte records before being sent to the AS/400 System. Other links allow the source data to remain in 400-byte records. However, the CRTGSS command will accept source data with record lengths in the range of 80 through 400 bytes. Symbol set source data with a record length of 400 can be sent back to the System/370 computer for modifications, if necessary, while 80-byte data needs to be blocked again before the symbol editors will accept it.

Symbol set source data can be placed into a physical file of either type *DATA or *SRC. Type *SRC data cannot be edited or viewed, and the record length of the file must be 12 bytes longer than the source data placed into it, to allow for the 12-byte sequence number fields associated with *SRC files. Therefore, the allowable record length range is 92 through 412 bytes for type *SRC files.

Once the symbol set source data is contained as a member in an AS/400 database file, it can be used to create a symbol set object (object type *GSS) in the CRTGSS command.

The database file source is provided by the System/370 GDDM-PGF Vector Symbol Editor in the following format (the format of the Image Symbol Editor database file is the same):



Graphics Symbol Set (*GSS) Objects

On the AS/400 System, graphics symbol sets are stored in the graphics symbol set object. The GSS object is an AS/400 object with an external symbolic type of *GSS.

Graphics Symbol Set CL Commands: You can use CL commands to tailor usage of the graphics symbol set objects to your own installation. Graphics symbol sets are supplied by IBM.

To manage graphics symbol sets use the CL object-management commands:

CRTGSS	Create Graphics Symbol Set
DLTGSS	Delete Graphics Symbol Set
RNMOBJ	Rename Object
MOVOBJ	Move Object
CRTDUPOBJ	Create Duplicate Object
SAVOBJ	Save Object
SAVCHGOBJ	Save Changed Object
RSTOBJ	Restore Object
GRTOBJAUT	Grant Object Authority
RVKOBJAUT	Revoke Object Authority
DSPOBJAUT	Display Object Authority
CHGOBJOWN	Change Object Ownership
DMPSYSOBJ	Dump System Object
DMPOBJ	Dump Object
WRKOBJLCK	Work with Object Locks

These CL object management commands cannot be used for graphics symbol sets:

ALCOBJ Allocate Object (lock)
DLCOBJ Deallocate Object (unlock).

You can issue the DSPOBJD command to see which symbol sets are available on your system.

For a complete description of any of these CL commands, refer to the *CL Reference* manual.

Performance Considerations

Some of your graphics applications programs could be programs that produce a simple picture or chart, others could be performing complex input/output operations and showing various pictures, while others draw highly-detailed charts.

If your program will be a complex one that other users execute, you should try to ensure that the processing time needed to construct each picture is acceptable. The more complex the picture, the more time is needed by the system to construct it.

Here are some ways to improve the performance of OS/400 Graphics application programs if response time is important:

- Avoid the use of curved lines in the pictures or charts.

  The system draws curves with many small, straight lines, whose end points must be defined (just like you define the end points of lines with GDDM). A circle with a diameter of 2.54 centimeters (one inch) is made up of more than 40 short lines.

  If curved lines are required on a chart, use a smaller number for the degree of smoothness for the CHFINE routine. Higher degrees of smoothness (larger numbers) might not be noticeable but will substantially increase the time needed to process the chart.

  Line or surface charts that use line curving, and Venn diagrams and pie charts that use circles are more complex for the system to construct than other types of chart.

- Use image symbols or the default vector symbol set ADMMVSS for graphics characters until the chart is in its final form, or use the default hardware characters. (The characters put on the screen by a display file are the fastest, most efficient way to show text.) Like arcs, each character is drawn with small lines. Complex fonts (such as Triplex) can multiply the number of lines the system must draw by factors of 10 through 100, depending on the number of characters your program uses.

- Use outlines instead of area-fills and shading patterns (these are also defined by the system with individual lines) for previewing pictures; later, you can add the patterns when the chart is in its final form.

  Sparse shading patterns are faster to draw than dense ones, especially on the plotter.

- Use a solid area-fill instead of a GDDM pattern for areas. When the area-fill pattern is solid, an area is enclosed by a maximum of 128 lines with boundary lines drawn, no two lines crossing, and no GSMOVE routines are used within the area, the graphics work station performs the area-fill, rather than the AS/400 System. (The AS/400 system must process the end points of each line in a pattern.)

- When your graphics program uses DDS display files, use FSFRCE to send the graphics output to the screen first, then use the display file information. This reduces the number of screen saves and restores.
- Use the Presentation Graphics routine CHTERM in the program as soon as you know that no more Presentation Graphics routines will be called. CHTERM releases the internal storage used by the program to process and construct the chart.
- When many continuous lines have to be drawn, GSPLNE is faster than several GSLINE calls.
- Structure a program that draws many pictures or charts such that you define all the pages before you select and display them at the end of the program. While this technique does not reduce processing time, it gives the illusion of increased performance by enabling you to display in quick succession a large number of pages whose processing time has already occurred (the pages are held in an internal form after they have been processed; they can be quickly selected and displayed).
- If your GDDM or Presentation Graphics program only draws a picture and shows it, (the picture does not have to be changed often), consider using the GDDM GDF routines to capture the processed graphic data format file. You can then display or plot the picture without having to execute the drawing routines each time.

Error Recovery

When GDDM detects an error in an application program, it sends a diagnostic message describing the error to the program message queue of the application program and creates an error record that contains the error message data.

Error-Handling Considerations

You can use the GDDM routine FSEXIT to specify an error-handling program that your GDDM or Presentation Graphics application program will call when an error is encountered.

If the error severity equals or exceeds the threshold specified in the FSEXIT routine, your error-handling program is invoked and the error record is passed to it. If FSEXIT is not specified and the severity of the error exceeds 40, or if FSEXIT is specified with only a severity threshold (and not the name of an error-handling program) whose value is equaled or exceeded by the error, the IBM-supplied default error message CPF8619 is issued.

If a threshold level of zero is specified, the error-handling program is invoked after *each* call to a GDDM routine, whether or not an error has occurred. In such cases, each error record contains only information about the most recent GDDM routine called.

A *default* error-handling program is set up on initialization, with a default threshold of 40 (unrecoverable error). This default error-handling program sends a CPF8619 escape message to the application program, and the application program is terminated (unless you are using the MONMSG (Monitor Message) command to perform other actions when the message is encountered).

If FSEXIT specifies the name of an error-handling program (in addition to the default), the program can be reset by the following statements:

```
CALL GDDM ('FSEXIT','*NONE',SEV)   ! SEV = severity
        or
CALL GDDM ('FSEXIT','*SAME',SEV)   ! SEV = severity
```

An error-handling program specified by FSEXIT is called by GDDM exactly as if it had been invoked instead of the GDDM routine that produced the error. The application program can use the error-handling program to call other GDDM routines, which are valid and current when control is returned to the original program (the one with the error).

If a file error occurs (perhaps due to a device failure), GDDM generates a severity 40 (unrecoverable) error. The application program must recover the device by using DSCLS followed by DSOPEN to close and reopen the file. The error message will show what you need to do.

The error record created by GDDM for the most recent error whose severity is nonzero can be returned to the application program by a call to FSQERR. The structure of the error record returned is identical to that passed to an error-handling program specified by FSEXIT. The error record is set to blanks in character fields and zeros in numeric fields before each call to FSQERR.

Error Messages

Message Identifiers: OS/400 Graphics *diagnostic* messages have a prefix of "CPG." All CPG messages are diagnostic messages; they further define an error condition or provide explanatory information.

OS/400 Graphics *escape* messages are CPF8600-series messages. If the message is unmonitored, termination occurs.

Further information regarding graphics and OS/400 messages is available through message help text.

Error Record Structure: The GDDM error record is similar to a record resulting from a RCVMSG (Receive Message) command.

The format of the error record is:

1. Severity. A number shows the AS/400 message severity:

   00 No error
   10 Warning
   20 Error (GDDM or Presentation Graphics routine is ignored)
   30 Severe error (unpredictable results may occur)
   40 Unrecoverable error.

2. Message identifier. The 7-character message identifier (CPGnnnn), where nnnn is the message number.

3. Routine name. Eight characters that contain the name of the GDDM routine that caused the error.

   For FSQERR, this field contains blanks if no error has occurred since the last call to FSQERR, or if there have been no errors since FSINIT.

4. Message text length. A 5-digit decimal number containing the length of the first-level message text, excluding trailing blanks. The maximum text length is 132.

5. Message text. A character field containing the first-level text of the error message associated with the error. 132 bytes are reserved in the error record structure for the message text, regardless of the length specified for the text.
6. RCP (request control parameter) code. A 4-byte binary integer with the RCP code of the GDDM routine. For FSQERR, the number is zero if no error has occurred since the last call to FSQERR or if there have been no errors since FSINIT.
7. Message reference key. The 4-character AS/400 message reference key of the message. This key is needed if the error-handling program receives the diagnostic message for any reason (with the RCVMSG (Receive Message) CL command).
8. Message type code. A 2-character field containing the AS/400 message type code for the message. This field usually contains 02 to indicate a *diagnostic* message. (The discussion for the RTNTYPE parameter of the RCVMSG command in the *CL Reference* manual shows other codes.)
9. Message data length. A 5-digit decimal number containing the length of the message data, if any. The minimum length is 12 (to accommodate the message variables listed below); the maximum length is 400.
10. Message data. The message data contains the substitution values (in a single character string) that were used in the text of the message. The amount of data returned and its format depend on the message.

All CPG-prefix messages have a format common to the first message variable, as follows:

1. Routine name. Eight characters that contain the name of the GDDM routine that caused the error. If the error exit threshold is zero or less, the routine name is that of the GDDM routine called. For FSQERR, the field contains blanks if no error has occurred since the last call to FSQERR or if there have been no errors since FSINIT.
2. Statement number. A 10-byte character string that identifies the statement in the program that caused the error.
3. The remainder of the message replacement variables.

User-Defined Data Streams

You can use a UDDS (user-defined data stream) with a user-defined keyword (USRDFN) to access the graphics work station and its attached devices for your applications, but it is not recommended.

If UDDS is used, the application program must be aware of the unusual nature of the graphics work station data stream and its *pacing* protocol for graphics. (Pacing is a method of communication between the graphics work station and the AS/400 System; pacing for the graphics work station differs from SNA pacing.)

You should be aware of these points before you use a UDDS:

- The graphics data stream must be sent in graphics blocks that must exceed 11 bytes but be less than or equal to 1920 bytes in length. More than one graphics block may be required to display a complete picture.
- Graphics blocks cannot contain normal alphanumeric data. All normal alphanumeric operations must be transmitted separately, and normal device rules apply to them.
- The graphics data stream is encoded such that all characters are in the range X'40' through X'FF'.
- All graphics input/output must be done as PUTGET operations to handle pacing for the display device, or pacing must be suppressed. The pacing scheme works as follows:
  1. The system sends a graphics block to the display device. The graphics block is recognized by the presence of X'FF' character immediately following the first WTD (write-to-display) order in the data stream. This causes the device to set graphics mode on and place the keyboard in *graphics lock state*, which is identified by a special blue input inhibited indicator on the screen. During graphics lock state, *no* keys can be pressed (including System Request) except for the special display device local keys.
  2. The display device processes the graphics block. Any block received during processing is interpreted as a normal alphanumeric block.
  3. When the display device finishes processing the graphics block (which may take some time, and can be minutes if the printer or plotter is being used), it sends back an aid key (F1 through F24) to indicate that it is ready for more work and that another graphics block may be sent down. The specific aid key indicates whether the processing completed normally or an error occurred.
  4. The system must wait for this aid key to be sent back before sending any more data to the display device.
  5. At the end of the last graphics block, the system must send an *end graphics mode* graphics order to signal to the display device to leave graphics mode. Otherwise, the keyboard will remain in graphics lock state. The only option you have is to use of one of the local keys on the display device to terminate graphics mode manually.
- If an event causes an interrupt (such as a break message or a system request) while the display device is in graphics mode (the application is in the process of transmitting one or more graphics blocks to the device), the application could be suspended, with an input operation that cannot be completed. This occurs when the normal graphics work station pacing response to a graphics block (the special aid key) is discarded by the work station controller because alphanumeric input/output is attempted while the display station is processing graphics. For more information on user-defined data streams, refer to the *IBM 5250 Information Display Systems Functions Reference Manual*.

Chapter 6. Graphics Application Program Examples

This chapter shows examples of programs in all the high-level languages that support OS/400 Graphics: BASIC, COBOL/400, Pascal, PL/I, and RPG/400 programming languages.

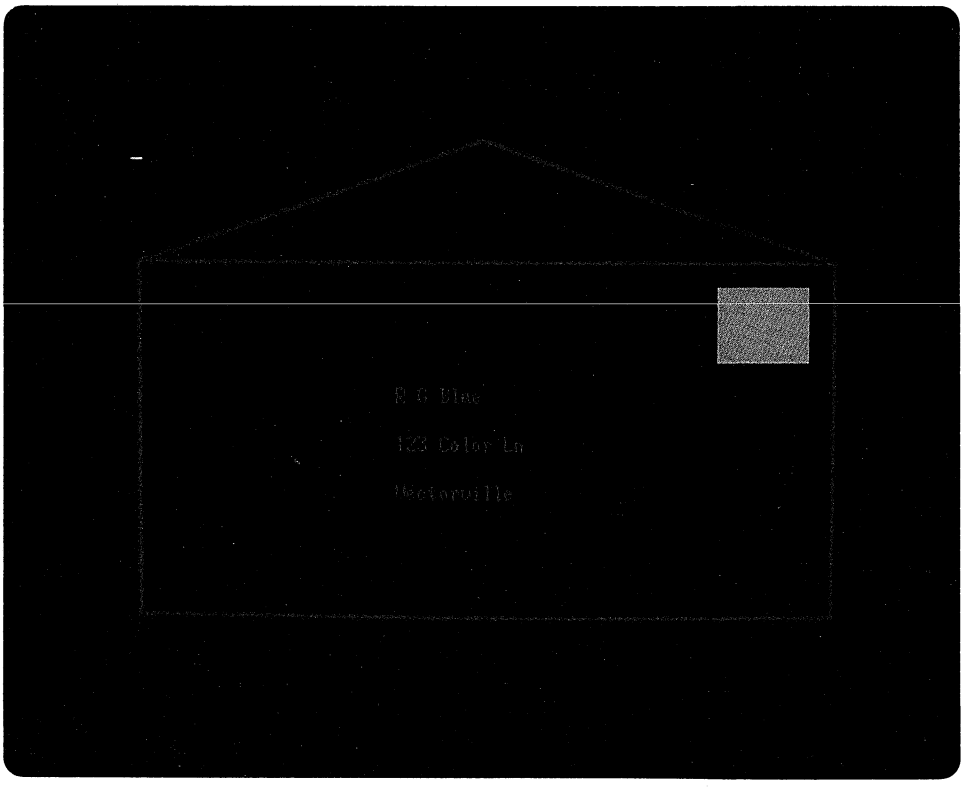
In Chapter 2, "The Application Program Interface to Graphics," two BASIC programs showed you the fundamentals of and the similarities and differences between GDDM and Presentation Graphics programs. The GDDM program drew a picture of a stamped, addressed envelope, and the Presentation Graphics program drew a line chart. The first part of this chapter shows the same programs in the other high-level languages.

The second part of this chapter shows other, more complex programs.

Finally there are examples of handling graphics images in each of the languages.

The Envelope Program in Other Languages

Each of the following programs provides source code, in one of the supported high-level languages, that will allow the following picture to be drawn:



© Copyright IBM Corp. 1991

Envelope Program in the RPG/400 Programming Language

```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     * The following three character variables read their string values
     * from the array input records found at the bottom of this
     * listing.
    E                    ADDR1   1   1  9              line 1 of address
    E                    ADDR2   1   1 12              line 2 of address
    E                    ADDR3   1   1 11              line 3 of address
    IPARAM       DS
    I                                    B   1   40LINEW
    I                                    B   5   80STRLEN
    I                                    B   9  120COLOR
    I                                    B  13  160OUTL
    I                                    B  17  200ATTYPE
    I                                    B  21  240ATMOD
    I                                    B  25  280COUNT
     ***************************************************************
     *                      INITIALIZE
     ***************************************************************
    C                    CALL 'GDDM'
    C                    PARM 'FSINIT  'FSINIT  8
     ***************************************************************
     *                     SET ATTRIBUTES
     ***************************************************************
     *            Assign line width (2 = wide)
    C                    CALL 'GDDM'
    C                    PARM 'GSLW    'GSLW    8
    C                    PARM 2         LINEW
     *            Assign color (5 = turquoise)
    C                    CALL 'GDDM'
    C                    PARM 'GSCOL   'GSCOL   8
    C                    PARM 5         COLOR
     ***************************************************************
     *                     DRAW ENVELOPE
     ***************************************************************
     *            Move to upper left corner
    C                    CALL 'GDDM'
    C                    PARM 'GSMOVE  'GSMOVE  8
    C                    PARM 1         X       51
    C                    PARM 75        Y       51
     *            Draw across to upper right
    C                    CALL 'GDDM'
    C                    PARM 'GSLINE  'GSLINE  8
    C                    PARM 80        X
    C                    PARM 75        Y
     *            Draw down to lower right
    C                    CALL 'GDDM'
    C                    PARM           GSLINE
    C                    PARM 80        X
    C                    PARM 1         Y
     *            Draw across to lower left
    C                    CALL 'GDDM'
    C                    PARM           GSLINE
    C                    PARM 1         X
    C                    PARM 1         Y
     *            Draw up to upper left corner
    C                    CALL 'GDDM'
    C                    PARM           GSLINE
```

```
C                   PARM 1          X
C                   PARM 75         Y
 *            Draw up to point of flap
C                   CALL 'GDDM'
C                   PARM            GSLINE
C                   PARM 40         X
C                   PARM 100        Y
 *            Draw down, over to upper right
C                   CALL 'GDDM'
C                   PARM            GSLINE
C                   PARM 80         X
C                   PARM 75         Y
 ************************************************************
 *               RESET ATTRIBUTES & DRAW STAMP
 ************************************************************
 *            Assign color (2 = red)
C                   CALL 'GDDM'
C                   PARM            GSCOL
C                   PARM 2          COLOR
 *            Specify filled area with outline
C                   CALL 'GDDM'
C                   PARM 'GSAREA  'GSAREA  8
C                   PARM 1          OUTL
 *            Move to upper left corner
C                   CALL 'GDDM'
C                   PARM 'GSMOVE  'GSMOVE  8
C                   PARM 67         X
C                   PARM 70         Y
 *            Draw across to upper right
C                   CALL 'GDDM'
C                   PARM            GSLINE
C                   PARM 77         X
C                   PARM 70         Y
 *            Draw down to lower right
C                   CALL 'GDDM'
C                   PARM            GSLINE
C                   PARM 77         X
C                   PARM 55         Y
 *            Draw across to lower left
C                   CALL 'GDDM'
C                   PARM            GSLINE
C                   PARM 67         X
C                   PARM 55         Y
 *            Draw up to upper left
C                   CALL 'GDDM'
C                   PARM            GSLINE
C                   PARM 67         X
C                   PARM 70         Y
 *            Fill the area
C                   CALL 'GDDM'
C                   PARM 'GSENDA  'GSENDA  8
 ************************************************************
 *                    WRITE ADDRESS
 ************************************************************
 *            Assign color (4 = green)
C                   CALL 'GDDM'
C                   PARM            GSCOL
C                   PARM 4          COLOR
 *            Write line 1
C                   CALL 'GDDM'
```

```
C                   PARM 'GSCHAR  'GSCHAR  8
C                   PARM 30        X
C                   PARM 45        Y
C                   PARM 9         STRLEN
C                   PARM           ADDR1
 *            Write line 2
C                   CALL 'GDDM'
C                   PARM           GSCHAR
C                   PARM 30        X
C                   PARM 35        Y
C                   PARM 12        STRLEN
C                   PARM           ADDR2
 *            Write line 3
C                   CALL 'GDDM'
C                   PARM           GSCHAR
C                   PARM 30        X
C                   PARM 25        Y
C                   PARM 11        STRLEN
C                   PARM           ADDR3
 ****************************************************************
 *                  DISPLAY THE PICTURE
 ****************************************************************
C                   CALL 'GDDM'
C                   PARM 'ASREAD  'ASREAD  8
C                   PARM 0         ATTYPE
C                   PARM           ATMOD
C                   PARM           COUNT
 ****************************************************************
 *                    TERMINATE
 ****************************************************************
C                   CALL 'GDDM'
C                   PARM 'FSTERM  'FSTERM  8
C                   SETON                    LR
C                   RETRN
 *
 * Note that SEU signals a syntax error when you enter
 * the following array input records; ignore the error.
**
R G Blue
**
123 Color Ln
**
Vectorville
```

Envelope Program in the COBOL/400 Programming Language

```
-A+++B++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 IDENTIFICATION DIVISION.
 PROGRAM-ID. ENVELOPE.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. IBM-S38.
 OBJECT-COMPUTER. IBM-S38.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  COLOR             PIC S9(5)    COMP-4.
 77  X                 PIC S9(5)    COMP-3.
 77  Y                 PIC S9(5)    COMP-3.
 77  ATTYPE            PIC S9(5)    COMP-4.
 77  ATMOD             PIC S9(5)    COMP-4.
 77  KOUNT             PIC S9(5)    COMP-4.
 77  STRING-LENGTH     PIC S9(5)    COMP-4.
 77  LINEWIDTH         PIC S9(5)    COMP-4.
 77  OUTLINES          PIC S9(5)    COMP-4   VALUE 0.
 01  GDDM-ROUTINES.
     05  GSCOL         PIC X(8)     VALUE "GSCOL".
     05  GSLW          PIC X(8)     VALUE "GSLW".
     05  GSMOVE        PIC X(8)     VALUE "GSMOVE".
     05  GSLINE        PIC X(8)     VALUE "GSLINE".
     05  GSAREA        PIC X(8)     VALUE "GSAREA".
     05  GSENDA        PIC X(8)     VALUE "GSENDA".
     05  GSCHAR        PIC X(8)     VALUE "GSCHAR".
     05  FSINIT        PIC X(8)     VALUE "FSINIT".
     05  FSTERM        PIC X(8)     VALUE "FSTERM".
     05  ASREAD        PIC X(8)     VALUE "ASREAD".
 01  ADDRESS-LINES.
     05  LINE1         PIC X(12)    VALUE "R G Blue    ".
     05  LINE2         PIC X(12)    VALUE "123 Color Ln".
     05  LINE3         PIC X(12)    VALUE "Vectorville ".
 PROCEDURE DIVISION.
 MAIN-ROUTINE.
     PERFORM TEST-PARAGRAPH.
 TEST-PARAGRAPH.
*****************************************************************
*                      INITIALIZE
*****************************************************************
     CALL "GDDM" USING FSINIT.
*****************************************************************
*                    SET ATTRIBUTES
*****************************************************************
* Assign line width (2 = wide)
     MOVE 2 TO LINEWIDTH.
     CALL "GDDM" USING GSLW, LINEWIDTH.
* Assign color (5 = turquoise)
     MOVE 5 TO COLOR.
     CALL "GDDM" USING GSCOL, COLOR.
*****************************************************************
*                     DRAW ENVELOPE
*****************************************************************
* Move to upper left corner
     MOVE 1 TO X.
     MOVE 75 TO Y.
```

```
     CALL "GDDM" USING GSMOVE, X, Y.
* Draw across to upper right
     MOVE 80 TO X.
     CALL "GDDM" USING GSLINE, X, Y.
* Draw down to lower right
     MOVE 1 TO Y.
     CALL "GDDM" USING GSLINE, X, Y.
* Draw across to lower left
     MOVE 1 TO X.
     CALL "GDDM" USING GSLINE, X, Y.
* Draw up to upper left corner
     MOVE 75 TO Y.
     CALL "GDDM" USING GSLINE, X, Y.
* Draw up to point of flap
     MOVE 40 TO X.
     MOVE 100 TO Y.
     CALL "GDDM" USING GSLINE, X, Y.
* Draw down, over to upper right
     MOVE 80 TO X.
     MOVE 75 TO Y.
     CALL "GDDM" USING GSLINE, X, Y.
*****************************************************************
*               RESET ATTRIBUTES & DRAW STAMP
*****************************************************************
* Assign color (2 = red)
     MOVE 2 TO COLOR.
     CALL "GDDM" USING GSCOL, COLOR.
* Specify filled area with outline
     MOVE 1 TO OUTLINES.
     CALL "GDDM" USING GSAREA, OUTLINES.
* Move to upper left corner
     MOVE 67 TO X.
     MOVE 70 TO Y.
     CALL "GDDM" USING GSMOVE, X, Y.
* Draw across to upper right
     MOVE 77 TO X.
     CALL "GDDM" USING GSLINE, X, Y.
* Draw down to lower right
     MOVE 55 TO Y.
     CALL "GDDM" USING GSLINE, X, Y.
* Draw across to lower left
     MOVE 67 TO X.
     CALL "GDDM" USING GSLINE, X, Y.
* Draw up to upper left
     MOVE 70 TO Y.
     CALL "GDDM" USING GSLINE, X, Y.
* Fill the area
     CALL "GDDM" USING GSENDA.
*****************************************************************
*                    WRITE ADDRESS
*****************************************************************
* Assign color (4 = green)
     MOVE 4 TO COLOR.
     CALL "GDDM" USING GSCOL, COLOR.
* Write line 1
     MOVE 12 TO STRING-LENGTH.
     MOVE 30 TO X.
     MOVE 45 TO Y.
     CALL "GDDM" USING GSCHAR, X, Y, STRING-LENGTH, LINE1.
* Write line 2
```

```
     MOVE 35 TO Y.
     CALL "GDDM" USING GSCHAR, X, Y, STRING-LENGTH, LINE2.
* Write line 3
     MOVE 25 TO Y.
     CALL "GDDM" USING GSCHAR, X, Y, STRING-LENGTH, LINE3.
*****************************************************************
*                    DISPLAY THE PICTURE
*****************************************************************
     CALL "GDDM" USING ASREAD, ATTYPE, ATMOD, KOUNT.
*****************************************************************
*                      TERMINATE
*****************************************************************
     CALL "GDDM" USING FSTERM.
     STOP RUN.
```

Envelope Program in PL/I

```
ENVELOPE: PROC;
DCL (ATTYPE,ATTVAL,COUNT) FIXED BIN(31);  /* Parameters for ASREAD */
/*********************************************************************/
/*                       INITIALIZE                                  */
/*********************************************************************/
CALL FSINIT;              /* Initialize the graphics environment    */
/*********************************************************************/
/*                     SET ATTRIBUTES                                */
/*********************************************************************/
CALL GSLW(2);             /* Assign line width (2 = wide)           */
CALL GSCOL(5);            /* Assign color (5 = turquoise)           */
/*********************************************************************/
/*                     DRAW ENVELOPE                                 */
/*********************************************************************/
CALL GSMOVE(1,75);        /* Move to upper left corner              */
CALL GSLINE(80,75);       /* Draw across to upper right             */
CALL GSLINE(80,1);        /* Draw down to lower left                */
CALL GSLINE(1,1);         /* Draw across to lower left              */
CALL GSLINE(1,75);        /* Draw up to upper left corner           */
CALL GSLINE(40,100);      /* Draw up to point of flap               */
CALL GSLINE(80,75);       /* Draw down, over to upper right         */
/*********************************************************************/
/*              RESET ATTRIBUTES & DRAW STAMP                        */
/*********************************************************************/
CALL GSCOL(2);            /* Assign color (2 = red)                 */
CALL GSAREA(1);           /* Specify filled area with outline       */
CALL GSMOVE(67,70);       /* Move to upper left corner              */
CALL GSLINE(77,70);       /* Draw across to upper right             */
CALL GSLINE(77,55);       /* Draw down to lower right               */
CALL GSLINE(67,55);       /* Draw across to lower left              */
CALL GSLINE(67,70);       /* Draw up to upper left                  */
CALL GSENDA;              /* Fill the area                          */
/*********************************************************************/
/*                      WRITE ADDRESS                                */
/*********************************************************************/
CALL GSCOL(4);                          /* Assign color (4 = green) */
CALL GSCHAR(30,45, 8,'R G Blue');       /* Write line 1             */
CALL GSCHAR(30,35,12,'123 Color Ln');   /* Write line 2             */
CALL GSCHAR(30,25,11,'Vectorville');    /* Write line 3             */
/*********************************************************************/
/*                  DISPLAY THE PICTURE                              */
/*********************************************************************/
CALL ASREAD(ATTYPE,ATTVAL,COUNT);       /* Display the picture      */
/*********************************************************************/
/*                       TERMINATE                                   */
/*********************************************************************/
CALL FSTERM;                            /* Terminate graphics       */
%INCLUDE SYSLIB (ADMUPLNB);             /* Include routine library  */
END ENVELOPE;                           /* End PL/I program         */
```

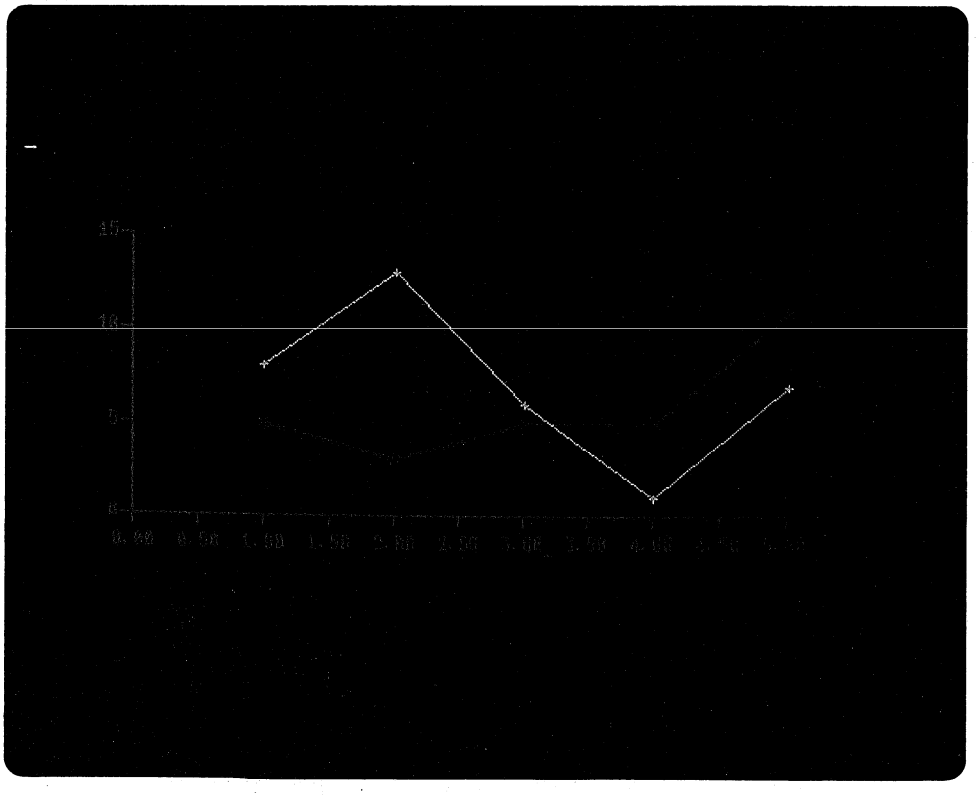
Envelope Program in Pascal

```
PROGRAM ENVELOPE;

TYPE
  %INCLUDE QATTPAS(ADMUSTNO);      /* IBM-supplied TYPE declarations  */

VAR
  X, Y : SHORTREAL;                /* work variables                  */
  CHARSTRING : CHARARR_132;        /* string parameter for GSCHAR     */
  A, B : INTEGER;                  /* work variables                  */
  ATTYPE, ATTVAL, COUNT : INTEGER; /* parameters for ASREAD           */

%INCLUDE QATTPAS(ADMUSLNB);        /* IBM-supplied PROC declarations  */

BEGIN
  /*********************************************************************/
  /*                          INITIALIZE                               */
  /*********************************************************************/
  FSINIT;                          /* Init the graphics environment   */
  /*********************************************************************/
  /*                       SET ATTRIBUTES                              */
  /*********************************************************************/
  A := 2;
  GSLW(A);                         /* Assign line width (2 = wide)    */
  B := 5;
  GSCOL(B);                        /* Assign color (5 = turquoise)    */
  /*********************************************************************/
  /*                       DRAW ENVELOPE                               */
  /*********************************************************************/
  X := 1.0; Y := 75.0;
  GSMOVE(X,Y);                     /* Move to upper left corner       */
  X := 80.0;
  GSLINE(X,Y);                     /* Draw across to upper right      */
  Y := 1.0;
  GSLINE(X,Y);                     /* Draw down to lower right        */
  X := 1.0;
  GSLINE(X,Y);                     /* Draw across to lower left       */
  Y := 75.0;
  GSLINE(X,Y);                     /* Draw up to upper left corner    */
  X := 40.0; Y := 100.0;
  GSLINE(X,Y);                     /* Draw up to point of flap        */
  X := 80.0; Y := 75.0;
  GSLINE(X,Y);                     /* Draw down, over to upper right  */
  /*********************************************************************/
  /*              RESET ATTRIBUTES AND DRAW STAMP                      */
  /*********************************************************************/
  A := 2;
  GSCOL(A);                        /* Assign color (2 = red)          */
  B := 1;
  GSAREA(B);                       /* Specify filled area with outline*/
  X := 67.0; Y := 70.0;
  GSMOVE(X,Y);                     /* Move to upper left corner       */
  X := 77.0;
  GSLINE(X,Y);                     /* Draw across to upper right      */
  Y := 55.0;
  GSLINE(X,Y);                     /* Draw down to lower right        */
  X := 67.0;
  GSLINE(X,Y);                     /* Draw across to lower left       */
  Y := 70.0;
```

```
  GSLINE(X,Y);                      /* Draw up to upper left          */
  GSENDA;                           /* Fill the area                  */
  /*********************************************************************/
  /*                      WRITE ADDRESS                                */
  /*********************************************************************/
  A := 4;
  GSCOL(A);                         /* Assign color (4 = green)       */
  X := 30.0; Y := 45.0; A := 8;
  CHARSTRING := 'R G Blue';
  GSCHAR(X,Y,A,CHARSTRING);         /* Write line 1                   */
  Y := 35.0; A := 12;
  CHARSTRING := '123 Color Ln';
  GSCHAR(X,Y,A,CHARSTRING);         /* Write line 2                   */
  Y := 25.0; A := 11;
  CHARSTRING := 'Vectorville';
  GSCHAR(X,Y,A,CHARSTRING);         /* Write line 3                   */
  /*********************************************************************/
  /*                    DISPLAY THE PICTURE                            */
  /*********************************************************************/
  ASREAD(ATTVAL,ATTYPE,COUNT);      /* Display the picture            */
  /*********************************************************************/
  /*                      END GRAPHICS                                 */
  /*********************************************************************/
  FSTERM                            /* End graphics                   */
END.                                /* End Pascal program             */
```

The Line Chart Program in Other Languages

Each of the following programs provides source code, in one of the supported high-level languages, that will allow the following chart to be drawn:



Line Chart Program in the RPG/400 Programming Language

```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     E                    AX          5  5 0
     E                    AY         10  5 0
     IPARAM       DS
     I                                       B   1   40LINES
     I                                       B   5   80POINTS
     I                                       B   9  120ATTYPE
     I                                       B  13  160ATMOD
     I                                       B  17  200COUNT
     C                    Z-ADD1             AX,1
     C                    Z-ADD2             AX,2
     C                    Z-ADD3             AX,3
     C                    Z-ADD4             AX,4
     C                    Z-ADD5             AX,5
     C                    Z-ADD5             AY,1
     C                    Z-ADD3             AY,2
     C                    Z-ADD5             AY,3
     C                    Z-ADD5             AY,4
     C                    Z-ADD11            AY,5
     C                    Z-ADD8             AY,6
     C                    Z-ADD13            AY,7
     C                    Z-ADD6             AY,8
     C                    Z-ADD1             AY,9
     C                    Z-ADD7             AY,10
     C                    MOVEL'FSINIT'      FSINIT  8
     C                    MOVEL'CHPLOT'      CHPLOT  8
     C                    MOVEL'FSTERM'      FSTERM  8
     C                    MOVEL'ASREAD'      ASREAD  8
      ************************************************************
      *                      INITIALIZE
      ************************************************************
     C                    CALL 'GDDM'
     C                    PARM               FSINIT
      ************************************************************
      *                    DRAW THE CHART
      ************************************************************
     C                    CALL 'GDDM'
     C                    PARM               CHPLOT
     C                    PARM 2             LINES
     C                    PARM 5             POINTS
     C                    PARM               AX
     C                    PARM               AY
      ************************************************************
      *                    DISPLAY THE CHART
      ************************************************************
     C                    CALL 'GDDM'
     C                    PARM               ASREAD
     C                    PARM               ATTYPE
     C                    PARM               ATMOD
     C                    PARM               COUNT
      ************************************************************
      *                       END GRAPHICS
      ************************************************************
     C                    CALL 'GDDM'
     C                    PARM               FSTERM
     C                    SETON                        LR
     C                    RETRN
```

Line Chart Program in the COBOL/400 Programming Language

```
-A+++B++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CHART.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-S38.
 OBJECT-COMPUTER.  IBM-S38.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77  LINENUM           PIC S9(5) COMP-4.
 77  POINTS            PIC S9(5) COMP-4.
 77  ATTYPE            PIC S9(5) COMP-4.
 77  ATMOD             PIC S9(5) COMP-4.
 77  KOUNT             PIC S9(5) COMP-4.
 77  FSINIT            PIC X(8)     VALUE "FSINIT".
 77  CHPLOT            PIC X(8)     VALUE "CHPLOT".
 77  FSTERM            PIC X(8)     VALUE "FSTERM".
 77  ASREAD            PIC X(8)     VALUE "ASREAD".
 01  X-ARRAY.
     03  AX  OCCURS 5 TIMES PIC S9(5)V9 COMP-3.
 01  Y-ARRAY.
     03  AY  OCCURS 10 TIMES PIC S9(5)V9 COMP-3.
 PROCEDURE DIVISION.
 MAIN-ROUTINE.
     PERFORM TEST-PARAGRAPH.
 TEST-PARAGRAPH.
     MOVE 1 TO AX (1).
     MOVE 2 TO AX (2).
     MOVE 3 TO AX (3).
     MOVE 4 TO AX (4).
     MOVE 5 TO AX (5).
     MOVE 5 TO AY (1).
     MOVE 3 TO AY (2).
     MOVE 5 TO AY (3).
     MOVE 5 TO AY (4).
     MOVE 11 TO AY (5).
     MOVE 8 TO AY (6).
     MOVE 13 TO AY (7).
     MOVE 6 TO AY (8).
     MOVE 1 TO AY (9).
     MOVE 7 TO AY (10).
*************************************************************
*                      INITIALIZE
*************************************************************
     CALL "GDDM" USING FSINIT.
*************************************************************
*                    DRAW THE CHART
*************************************************************
     MOVE 2 TO LINENUM.
     MOVE 5 TO POINTS.
     CALL "GDDM" USING CHPLOT, LINENUM, POINTS, AX, AY.
```

```
************************************************************
*                    DISPLAY THE CHART
************************************************************
     CALL "GDDM" USING ASREAD, ATTYPE, ATMOD, KOUNT.
************************************************************
*                     END GRAPHICS
************************************************************
     CALL "GDDM" USING FSTERM.
     STOP RUN.
```

Line Chart Program in PL/I

```
CHARTPLI: PROC;
DCL
    (ATTYPE,ATTVAL,COUNT) FIXED BIN(31),   /* Parameters for ASREAD */
    COMPONENTS  FIXED BIN(31) STATIC INIT(2),
    POINTS  FIXED BIN(31) STATIC INIT(5),
    AX(5)  FLOAT DEC(6) STATIC INIT(1, 2, 3, 4, 5),
    AY(10) FLOAT DEC(6) STATIC INIT(5, 3, 5, 5, 11,
                                    8, 13, 6, 1, 7);
/*********************************************************************/
/*                        INITIALIZE                                 */
/*********************************************************************/
CALL FSINIT;
/*********************************************************************/
/*                       DRAW THE CHART                              */
/*********************************************************************/
CALL CHPLOT(COMPONENTS,POINTS,AX,AY);
/*********************************************************************/
/*                      DISPLAY THE CHART                            */
/*********************************************************************/
CALL ASREAD(ATTYPE,ATTVAL,COUNT);
/*********************************************************************/
/*                         END GRAPHICS                              */
/*********************************************************************/
CALL FSTERM;
%INCLUDE SYSLIB (ADMUPLNO);
END CHARTPLI;
```

Line Chart Program in Pascal

```
PROGRAM LINECHRT;

TYPE
  %INCLUDE QATTPAS(ADMUSTNO);      /* IBM-supplied TYPE declarations  */

VAR
  A, B : INTEGER;                  /* work variables                  */
  AX, AY : REALARR_20;             /* parameters for CHPLOT           */
  ATTVAL, ATTYPE, COUNT : INTEGER; /* parameters for ASREAD           */

%INCLUDE QATTPAS(ADMUSLNO);        /* IBM-supplied PROC declarations  */

BEGIN
```

```
/*****************************************************************/
/*                        INITIALIZE                             */
/*****************************************************************/
AX(.1.) := 1.0;  AY(.1.) := 5.0;  AY(.6.) := 8.0; /* Init arrays    */
AX(.2.) := 2.0;  AY(.2.) := 3.0;  AY(.7.) := 13.0;
AX(.3.) := 3.0;  AY(.3.) := 5.0;  AY(.8.) := 6.0;
AX(.4.) := 4.0;  AY(.4.) := 5.0;  AY(.9.) := 1.0;
AX(.5.) := 5.0;  AY(.5.) := 11.0; AY(.10.) := 7.0;
FSINIT;                          /* Initialize graphics environment */
/*****************************************************************/
/*                     DRAW THE CHART                            */
/*****************************************************************/
A := 2;
B := 5;
CHPLOT(A,B,AX,AY);               /* Draw the chart                 */
/*****************************************************************/
/*                    DISPLAY THE CHART                          */
/*****************************************************************/
ASREAD(ATTVAL,ATTYPE,COUNT);     /* Display the chart              */
/*****************************************************************/
/*                     END GRAPHICS                              */
/*****************************************************************/
FSTERM                           /* End graphics environment       */
END.                             /* End Pascal program             */
```

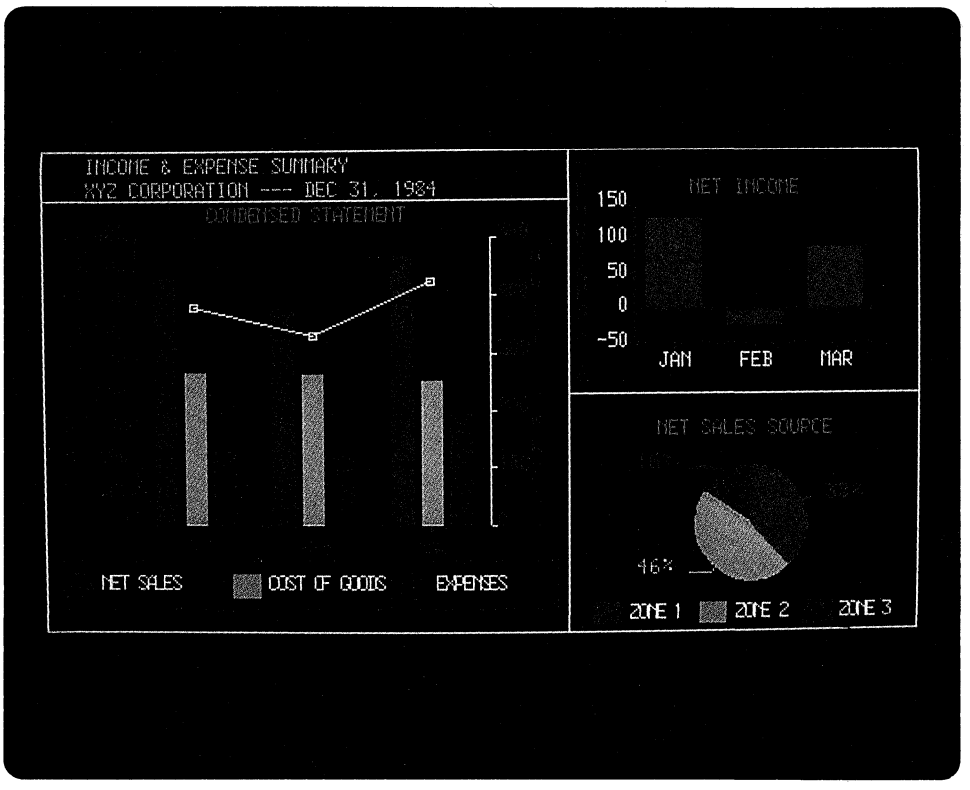
Complex Programs

The programs in this section show some of the many variations available for graphics application programs.

BASIC Program Showing Three Charts on a Page

This example program shows how to use:

- Data from a database file for chart data
- GDDM and Presentation Graphics in the same program
- A secondary axis on a chart
- Three separate charts placed in separate chart areas.



The combination of charts is drawn by this program:

```
00010 SALES: !
00020 !*****************************************************************
00030 !* This program displays an income and expense summary for the  *
00040 !* XYZ Corporation by charting a condensed statement, a net      *
00050 !* income chart, and a net sales chart.  All three charts are    *
00060 !* shown in the same picture.                                    *
00070 !*                                                               *
00080 !* Declare all arrays and variables used in the program:         *
00090 !*****************************************************************
00100    OPTION BASE 1
00110    DIM MULBAR(9),NETSAL(3),COST(3),EXPEN(3),NETINC(3),MONTH(3)
00120    DIM UNITS(3),COLTAB(4),PATTAB(4)
00130    INTEGER ATTYPE,ATMOD
00140    DIM HATTS(4),TATTS(4),LATTS(4),KATTS(4),VATTS(4),AATTS(12)
00150    INTEGER LENGTH,POSITIONS,AXIS,COMP
00160    INTEGER HATTS,TATTS,LATTS,KATTS,COUNT,STARTMON,MODE,COLOR
```

```
00170    INTEGER LMAR,RMAR,BMAR,TMAR,VATTS,AATTS,COLTAB,PATTAB
00180    DECIMAL WIDTH,DEPTH,X,Y,MULBAR,DLINE
00190    DECIMAL NETSAL,COST,EXPEN,NETINC,UNITS,NETSS
00200 !*************************************************************
00210 !* Initialize arrays for the heading, axes, axis titles,      *
00220 !* labels, and value text.                                    *
00230 !*************************************************************
00240    HATTS(1)=3  : HATTS(2)=2  : HATTS(3)=0  : HATTS(4)=160
00250    AATTS(1)=-1 : AATTS(2)=-1 : AATTS(3)=-1 : AATTS(4)=-1
00260    AATTS(5)=-1 : AATTS(6)=-1 : AATTS(7)=-1 : AATTS(8)=-1
00270    AATTS(9)=-1 : AATTS(10)=6 : AATTS(11)=0 : AATTS(12)=1
00280    TATTS(1)=4  : TATTS(2)=2  : TATTS(3)=0  : TATTS(4)=150
00290    LATTS(1)=1  : LATTS(2)=2  : LATTS(3)=0  : LATTS(4)=160
00300    KATTS(1)=6  : KATTS(2)=2  : KATTS(3)=0  : KATTS(4)=135
00310    VATTS(1)=5  : VATTS(2)=2  : VATTS(3)=0  : VATTS(4)=160
00320 !*************************************************************
00330 !* Open the file that contains the data used to draw the      *
00340 !* three charts.  The file could have been generated using the *
00350 !* Query utility with the output going to this data base file. *
00360 !* Read the data and calculate the net income, then store the  *
00370 !* net sales, cost, and expense figures in an array for later  *
00380 !* use in the bar chart.                                      *
00390 !*************************************************************
00400    OPEN #1 :'NAME=DBFILE,LIBRARY=YOURLIB',INPUT
00410    FOR I = 1 TO 3
00420       READ #1,USING'FORM ZD 3.1,5*ZD 6.2':MONTH(I),NETSAL(I),COS&
&T(I),EXPEN(I),UNITS(I),NETSS(I)
00430       NETINC(I) = NETSAL(I) - (COST(I) + EXPEN(I))
00440       MULBAR(I) = NETSAL(I) : MULBAR(I+3) = COST(I)
00450       MULBAR(I+6)=EXPEN(I)
00460    NEXT I
00470 !*************************************************************
00480 !* Initialize the graphics environment                        *
00490 !*************************************************************
00500    CALL GDDM('FSINIT')
00510 !*************************************************************
00520 !* Use subroutines to write the notes and draw the charts.    *
00530 !*************************************************************
00540    GOSUB NOTES
00550    GOSUB CONSTAT
00560    GOSUB NETINCOME
00570    GOSUB NETSSOURCE
00580 !*************************************************************
00590 !* Terminate Presentation Graphics, send the picture to the   *
00600 !* display, and terminate GDDM.                               *
00610 !*************************************************************
00620    CALL GDDM('CHTERM')
00630    CALL GDDM('ASREAD',ATTYPE,ATMOD,COUNT)
00640    CALL GDDM('FSTERM')
00650 GOTO DONE
00660 NOTES: !
00670 !*************************************************************
00680 !* GDDM SUBROUTINE TO WRITE NOTES IN THE UPPER LEFT CORNER    *
00690 !* OF THE PICTURE.                                            *
00700 !*                                                           *
00710 !* Set the color of the characters, set the character mode    *
00720 !* to two, set the character box size, then draw the string   *
00730 !* at position X,Y.                                           *
00740 !*************************************************************
00750    CALL GDDM('GSSEG',0)
```

```
00760    COLOR=2
00770    CALL GDDM('GSCOL',COLOR)
00780    X=5 : Y=95 : LENGTH=24
00790    CALL GDDM('GSCHAR',X,Y,LENGTH,'INCOME & EXPENSE SUMMARY')
00800    X=5 : Y=90.0 : LENGTH=15
00810    CALL GDDM('GSCHAR',X,Y,LENGTH,'XYZ CORPORATION')
00820    LENGTH=17
00830    CALL GDDM('GSCHAP',LENGTH,' --- DEC 31, 1984')
00840 !*****************************************************************
00850 !* Change the color, then draw the boundary lines for the note. *
00860 !*****************************************************************
00870    COLOR=7
00880    CALL GDDM('GSCOL',COLOR)
00890    X=0.0 : Y=88.0
00900    CALL GDDM('GSMOVE',X,Y)
00910    X=0.0 : Y=100.00
00920    CALL GDDM('GSLINE',X,Y)
00930    X=60.25 : Y=100.00
00940    CALL GDDM('GSLINE',X,Y)
00950    CALL GDDM('GSSCLS')
00960 RETURN
00970 CONSTAT: !
00980 !*****************************************************************
00990 !* SUBROUTINE TO DRAW THE CONDENSED STATEMENT CHART.             *
01000 !*                                                               *
01010 !* Query the picture space, then set the chart area and          *
01020 !* draw a frame around the chart.                                *
01030 !*****************************************************************
01040    CALL GDDM('GSQPS',WIDTH,DEPTH)
01050    CALL GDDM('CHAREA',0.0,.6*WIDTH,0.0,.9*DEPTH)
01060    CALL GDDM('CHSET','CBOX')
01070 !*****************************************************************
01080 !* Set the attributes for and write the heading.                 *
01090 !*****************************************************************
01100    POSITIONS=4
01110    CALL GDDM('CHHATT',POSITIONS,HATTS())
01120    LENGTH = 19
01130    CALL GDDM('CHHEAD',LENGTH,'CONDENSED STATEMENT')
01140 !*****************************************************************
01150 !* Set the attributes for the axis labels and axes, set the      *
01160 !* margins.                                                      *
01170 !*****************************************************************
01180    POSITIONS=4
01190    CALL GDDM('CHLATT',POSITIONS,LATTS())
01200    POSITIONS=12
01210    CALL GDDM('CHAATT',POSITIONS,AATTS())
01220    LMAR=14 : RMAR=12
01230    CALL GDDM('CHVMAR',LMAR,RMAR)
01240    BMAR=6 : TMAR=2
01250    CALL GDDM('CHHMAR',BMAR,TMAR)
01260 !*****************************************************************
01270 !* Position, set the attributes for, and draw the legend.        *
01280 !*****************************************************************
01290    CALL GDDM('CHKEYP','H','B','C')
01300    X=0.0 : Y= 1.3
01310    CALL GDDM('CHKOFF',X,Y)
01320    CALL GDDM('CHSET','KBOX')
01330    POSITIONS=4
01340    CALL GDDM('CHKATT',POSITIONS,KATTS())
01350    COUNT=3
```

```
01360    LENGTH=13
01370    CALL GDDM('CHKEY',COUNT,LENGTH,'NET SALES    COST OF GOODSEXP&
&ENSES     ')
01380 !*************************************************************
01390 !* Set the attributes for and write the axis titles.          *
01400 !*************************************************************
01410    POSITIONS=4
01420    CALL GDDM('CHTATT',POSITIONS,TATTS())
01430    LENGTH = 11
01440    CALL GDDM('CHYTTL',LENGTH,'THOUSANDS $')
01450 !*************************************************************
01460 !* Set the starting month for the x-axis labels, and set the  *
01470 !* color table and the pattern table.                         *
01480 !*************************************************************
01490    STARTMON=1
01500    CALL GDDM('CHXMTH',STARTMON)
01510    COLTAB(1)=1 : COLTAB(2)=2 : COLTAB(3)=5 : COLTAB(4)=6
01520    CALL GDDM('CHCOL',POSITIONS,COLTAB())
01530    PATTAB(1)=0 : PATTAB(2)=0 : PATTAB(3)=0 : PATTAB(4)=0
01540    CALL GDDM('CHPAT',POSITIONS,PATTAB())
01550 !*************************************************************
01560 !* Select and define the secondary axis used for the          *
01570 !* line chart that represents sales units.                    *
01580 !*************************************************************
01590    AXIS = 2
01600    CALL GDDM('CHYSEL',AXIS)
01610    LENGTH = 10
01620    CALL GDDM('CHYTTL',LENGTH,'UNIT SALES')
01630    X=0.0 : Y=500.00
01640    CALL GDDM('CHYRNG',X,Y)
01650    X=100.00 : Y=0.0
01660    CALL GDDM('CHYTIC',X,Y)
01670    CALL GDDM('CHYSET','PTICK')
01680 !*************************************************************
01690 !* Select the primary axis and draw the bar chart.            *
01700 !*************************************************************
01710    AXIS = 1
01720    CALL GDDM('CHYSEL',AXIS)
01730    COMP = 3 : COUNT=3
01740    CALL GDDM('CHBAR',COMP,COUNT,MULBAR())
01750 !*************************************************************
01760 !* Select the secondary axis, then draw the                   *
01770 !* line chart component that represents sales units.          *
01780 !*************************************************************
01790    AXIS = 2
01800    CALL GDDM('CHYSEL',AXIS)
01810    COMP = 1 : COUNT=3
01820    CALL GDDM('CHPLOT',COMP,COUNT,MONTH(),UNITS())
01830 !*************************************************************
01840 !* Draw the axes again to overpaint the bar chart components. *
01850 !*************************************************************
01860    CALL GDDM('CHDRAX')
01870 !*************************************************************
01880 !* Reset all chart definition values to default values and    *
01890 !* end the subroutine.                                        *
01900 !*************************************************************
01910    CALL GDDM('CHRNIT')
01920 RETURN
01930 NETINCOME:  !
01940 !*************************************************************
```

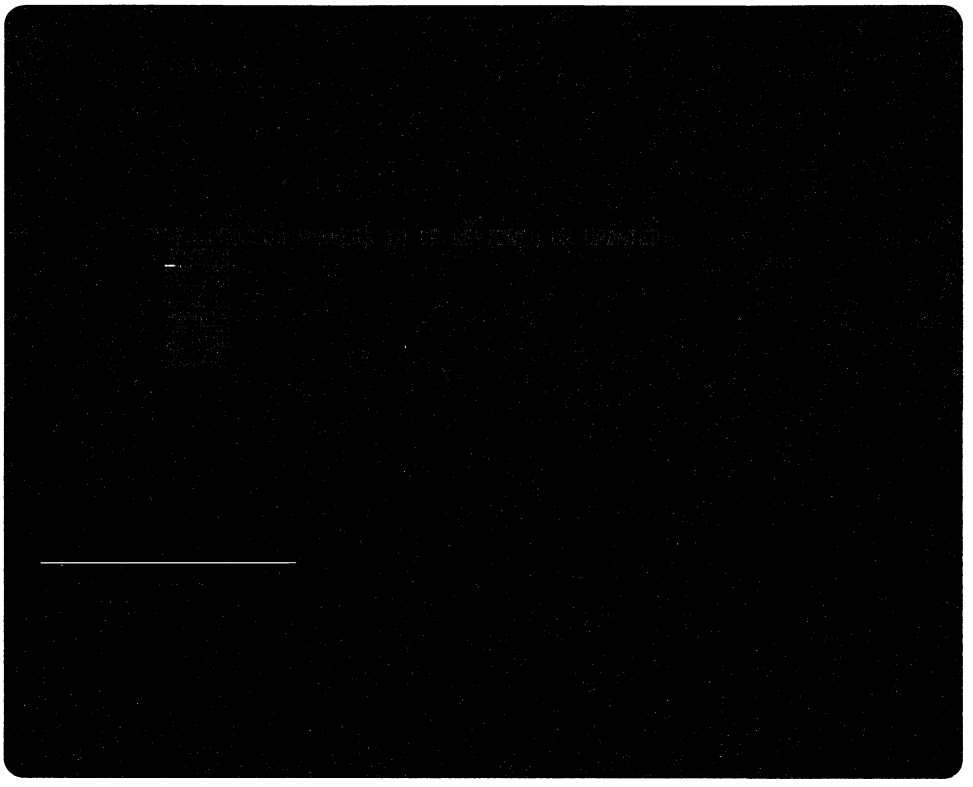
```
01950 !* SUBROUTINE TO DRAW THE NET INCOME CHART FOR CORPORATION XYZ. *
01960 !*                                                              *
01970 !* Query the picture space, then set the chart area and         *
01980 !* draw a frame around the chart.                               *
01990 !***************************************************************
02000    CALL GDDM('CHAREA',0.6*WIDTH,WIDTH,0.5*DEPTH,DEPTH)
02010    CALL GDDM('CHSET','CBOX')
02020    CALL GDDM('CHVMAR',15,10)
02030 !***************************************************************
02040 !* Set the attributes for and write the heading.  The heading   *
02050 !* is moved down from the boundary with the line-break ';'.     *
02060 !***************************************************************
02070    HATTS(4)=250
02080    CALL GDDM('CHHATT',POSITIONS,HATTS())
02090    LENGTH = 11
02100    CALL GDDM('CHHEAD',LENGTH,';NET INCOME')
02110    LATTS(1)=6 : LATTS(4)=250
02120    CALL GDDM('CHLATT',POSITIONS,LATTS())
02130 !***************************************************************
02140 !* Suppress the legend, set the starting month for the x-axis   *
02150 !* labels, specify a datum line for the y axis, set the         *
02160 !* attributes for the x- and y-axis labels, and write           *
02170 !* the y-axis title.                                            *
02180 !***************************************************************
02190    CALL GDDM('CHSET','NOLEGEND')
02200    CALL GDDM('CHXMTH',STARTMON)
02210    DLINE=0
02220    CALL GDDM('CHYDTM',DLINE)
02230    TATTS(4)=200
02240    CALL GDDM('CHTATT',POSITIONS,TATTS())
02250    LENGTH = 11
02260    CALL GDDM('CHYTTL',LENGTH,'THOUSANDS $')
02270 !***************************************************************
02280 !* Draw the net income bar chart.                               *
02290 !***************************************************************
02300    COMP = 1 : COUNT=3
02310    CALL GDDM('CHBAR',COMP,COUNT,NETINC())
02320 !***************************************************************
02330 !* Draw the axes again to overpaint the bar chart components.   *
02340 !***************************************************************
02350    CALL GDDM('CHDRAX')
02360 !***************************************************************
02370 !* Reset all chart definition values to default values and      *
02380 !* end the subroutine.                                          *
02390 !***************************************************************
02400    CALL GDDM('CHRNIT')
02410 RETURN
02420 NETSSOURCE:  !
02430 !***************************************************************
02440 !* SUBROUTINE TO DRAW THE NET SALES SOURCE CHART FOR XYZ.       *
02450 !*                                                              *
02460 !* Query the picture space, then set the chart area and         *
02470 !* draw a frame around the chart.                               *
02480 !***************************************************************
02490    CALL GDDM('CHAREA',0.6*WIDTH,WIDTH,0.0,0.5*DEPTH)
02500    CALL GDDM('CHSET','CBOX')
02510 !***************************************************************
02520 !* Set the attributes for and write the heading.                *
02530 !***************************************************************
02540    CALL GDDM('CHHATT',POSITIONS,HATTS())
```

```
02550    LENGTH = 17
02560    CALL GDDM('CHHEAD',LENGTH,';NET SALES SOURCE')
02570 !*************************************************************
02580 !* Use value text and absolute data for the chart.           *
02590 !*************************************************************
02600    CALL GDDM('CHSET','VALUES')
02610    CALL GDDM('CHSET','ABPIE')
02620 !*************************************************************
02630 !* Set the margins and the color and pattern tables.         *
02640 !*************************************************************
02650    CALL GDDM('CHVMAR',5,2)
02660    CALL GDDM('CHHMAR',3,3)
02670    CALL GDDM('CHCOL',POSITIONS,COLTAB())
02680    CALL GDDM('CHPAT',POSITIONS,PATTAB())
02690 !*************************************************************
02700 !* Position and draw the legend.                             *
02710 !*************************************************************
02720    CALL GDDM('CHKEYP','H','B','C')
02730    X=0.0 : Y=-1.6
02740    CALL GDDM('CHKOFF',X,Y)
02750    KATTS(2)=2 : KATTS(4)=200
02760    CALL GDDM('CHKATT',POSITIONS,KATTS())
02770    LENGTH=6
02780    CALL GDDM('CHKEY',COUNT,LENGTH,'ZONE 1ZONE 2ZONE 3')
02790 !*************************************************************
02800 !* Set the attributes for the labels and value text.         *
02810 !*************************************************************
02820    LATTS(4)=300
02830    CALL GDDM('CHLATT',POSITIONS,LATTS())
02840    VATTS(4)=300
02850    CALL GDDM('CHVATT',POSITIONS,VATTS())
02860 !*************************************************************
02870 !* Draw the net sales source pie chart.                      *
02880 !*************************************************************
02890    COMP = 1 : COUNT=3
02900    CALL GDDM('CHPIE',COMP,COUNT,NETSS())
02910 RETURN
02920 DONE: END SALES
```

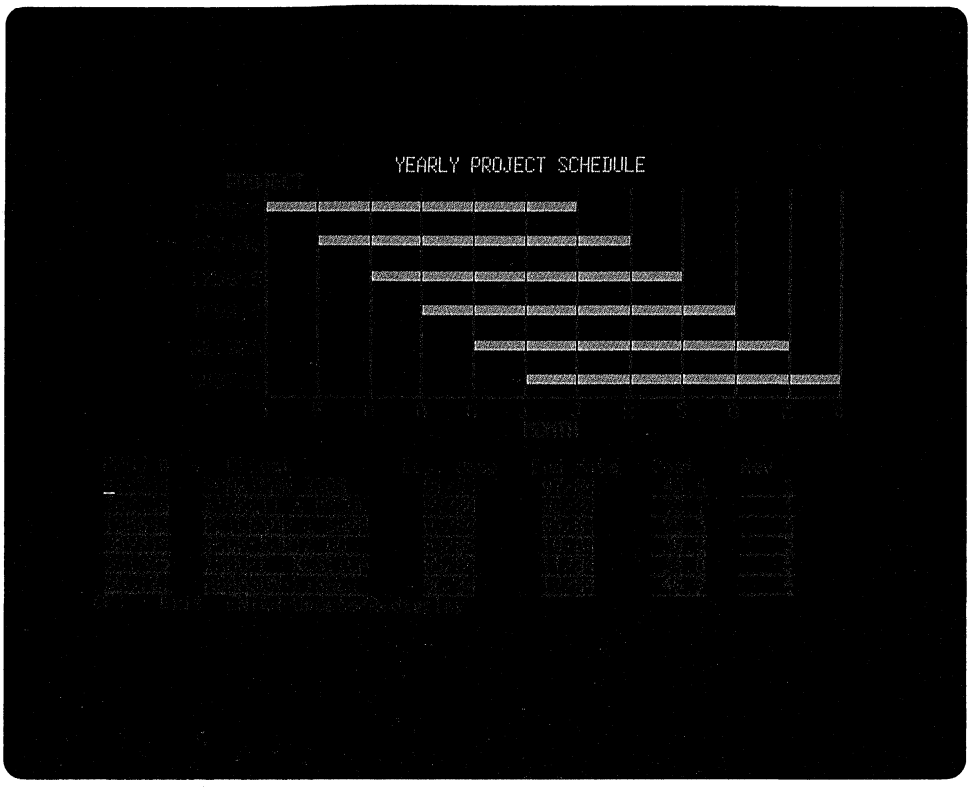
BASIC Program that Interacts with Database Files

This program is a project scheduling application that uses a two-subfile display file to read data from and write data to a database file. The data is used to draw a horizontal, floating, single-bar chart.

The first subfile produces the following display, which you can use to enter project numbers for data that already exists in the database file:



The second subfile works with the graphics program as shown in the following picture. The second subfile also accepts data entered for new projects.



There are two database files used for the project data. This is the *accounting* file called PACCT:

```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     A                                      UNIQUE
     A          R ACCT                      TEXT('Project Accounting File')
     A            PROJ          6A          TEXT('Project Number')
     A            COST          4  0        TEXT('Project Cost')
     A            REV           4  0        TEXT('Project Revenue')
     A          K PROJ
```

This is the *schedule* file called PSCHED:

```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     A                                      UNIQUE
     A          R SCHED                     TEXT('Project Schedule File')
     A            PROJ          6A          TEXT('Project Number')
     A            CLIENT       15A          TEXT('Client Name')
     A            BEGDAT        5A          TEXT('Beginning Date')
     A            ENDDAT        5A          TEXT('Ending Date')
     A          K PROJ
```

The following is the two-subfile display file called PROJDSP:

```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     A                                      DSPSIZ(*DS3)
     A                                      HELP(99 'HELP KEY')
     A                                      CA01(98 'EXIT THE APPLIC')
     A                                      ALWGPH
     A          R PROJ                      SFL
     A            PROJ           6A  I007003
     A          R PROJCTL                   SFLCTL(PROJ)
     A                                      TEXT('GRAPHICS APPLICATION  1
     A                                           ')
     A                                      SFLSIZ(06)
     A                                      SFLPAG(06)
     A  84                                  SFLEND
     A  85                                  SFLDSP
     A  84                                  SFLDSPCTL
     A  83                                  SFLCLR
     A  86                                  SFLINZ
     A                                   006001'ENTER PROJECT NUMBERS TO BE'
     A                                   006029'REVIEWED OR UPDATED:'
     A          R SFLA                      SFL
     A            PROJ           6A  B017003
     A            CLIENT        15A  B017012
     A            BEGDAT         5A  B017032
     A            ENDDAT         5A  B017043
     A            COST           4  0B017053EDTWRD('   . ')
     A            REV            4  0B017061EDTWRD('   . ')
     A          R SFCTL                     SFLCTL(SFLA)
     A                                      TEXT('GRAPHICS APPLICATION  1
     A                                           ')
     A                                      SFLSIZ(06)
     A                                      SFLPAG(06)
     A  84                                  SFLEND
     A  85                                  SFLDSP
     A  84                                  SFLDSPCTL
     A  83                                  SFLCLR
     A  86                                  SFLINZ
     A                                   016003'PROJ #'
     A                                   016014'Client'
     A                                   016030'Beg. date'
     A                                   016042'End date'
     A                                   016053'Cost'
     A                                   016061'Rev'
     A          R OVERL
     A                                      OVERLAY
     A                                   023002'CF1 - Exit  ENTER-Update/Redi-
     A                                         splay'
     A          R CLEAR
```

This is the graphics program that uses the two database files and the display file to show the project schedules based on the database file data:

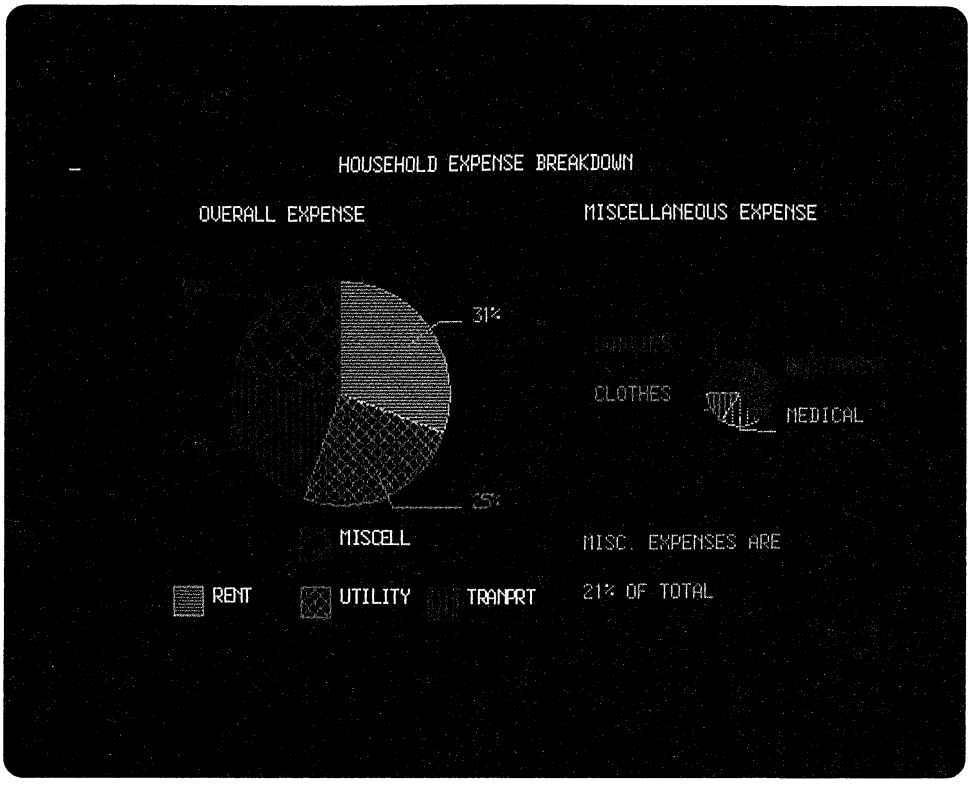
```
00010 !*************************************************************
00020 !*                  PROJECT SCHEDULE PROGRAM                 *
00030 !*                                                           *
00040 !* This program plots project schedules for up to 6 projects  *
00050 !* and list information about each project.                  *
00060 !*                                                           *
00070 !* Three files are used in this application:                 *
00080 !*     PACCT   : project accounting data base file           *
00090 !*     PSCHED  : project schedule data base file             *
00100 !*     PROJDSP : display file                                *
00110 !*                                                           *
00120 !*************************************************************
00130 OPTION BASE 1                                  ! Subscript base 1
00140 DIM BDATE$*5, EDATE$*5, INDARA$*99             ! Char variables
00150 DIM COLARRAY(2), AXSARRAY(6)                   ! Arrays
00160 DIM HDGARRAY(2), PATARRAY(1)                   ! Arrays
00170 DIM LBLARRAY(4), TTLARRAY(2), GRDARRAY(6)      ! Arrays
00180 DIM STARTARRAY(6),ENDARRAY(6), DATAARRAY(12)   ! Arrays
00190 DIM PROJLIST$(6)*6, XLABEL$*36                 ! Project list
00200 INDARA$ = RPT$('0',99)                         ! Indicator area
00210 INTEGER NUM, I                                 ! Integers
00220 INTEGER COLARRAY, AXSARRAY                     ! Integers
00230 INTEGER PATARRAY, HDGARRAY                     ! Integers
00240 INTEGER LBLARRAY, TTLARRAY, GRDARRAY           ! Integers
00250 !*************************************************************
00260 !* Declare and open the display file and the two data base files
00270 !*************************************************************
00280 DECLARE FILE #5: "PROJDSP.YOURLIB" EXTDESCR   ! Declare file
00290 DECLARE FILE #6: "PSCHED.YOURLIB" EXTDESCR    ! Declare file
00300 DECLARE FILE #7: "PACCT.YOURLIB" EXTDESCR     ! Declare file
00310 OPEN #5:"WS,NAME=PROJDSP,LIB=YOURLIB,FORMAT"  ! Open file
00320 OPEN #6:"FILE,NAME=PSCHED,FORMAT",KEYED, OUTIN ! Open file
00330 OPEN #7:"FILE,NAME=PACCT,FORMAT",KEYED, OUTIN  ! Open file
00340 !*************************************************************
00350 !* Request the input of up to 6 project numbers.
00360 !*************************************************************
00370 INDARA$(86:86) = "1"             ! Subfile initialize on
00380 INDARA$(84:84) = "1"             ! Subfile control display off
00390 INDARA$(85:85) = "1"             ! Subfile initialize on
00400 WRITE #5, EXTDESCR "PROJCTL", INDIC INDARA$:
00410 READ #5, EXTDESCR "PROJCTL":
00420 WRITE #5, EXTDESCR "CLEAR", INDIC INDARA$:  ! Clear screen
00430 NUM = 0
00440 FOR I = 1 TO 6
00450   READ #5, EXTDESCR "PROJ",REC=I:
00460   IF PROJ$ = "      " THEN GOTO 490
00470   NUM = NUM + 1
00480   PROJLIST$(NUM) = PROJ$
00490 NEXT I
00500 !*************************************************************
00510 !* Set the attributes for the chart.
00520 !*************************************************************
00530 GOSUB SETGRAPH
00540 !*************************************************************
00550 !*
00560 !* SUBROUTINE TO INITIALIZE THE DISPLAY FILE
00570 !*
```

```
00580 !************************************************************
00590 REDISPLAY:                        ! Loop to redisplay the chart
00600   INDARA$(84:85) = "00"           ! Subfile control display off
00610   INDARA$(86:86) = "1"            ! Subfile initialize on
00620   WRITE #5,EXTDESCR "SFCTL", INDIC INDARA$:
00630   !************************************************************
00640   !* Fill the subfile with the project information requested.
00650   !************************************************************
00660   FOR I = 1 TO NUM
00670     PROJ$ = PROJLIST$(I)
00680     GOSUB ADDSUBF
00690   NEXT I
00700   !************************************************************
00710   !* Set the subfile indicators and display the project data.
00720   !************************************************************
00730   INDARA$(86:86) = "0"            ! Subfile initialize off
00740   INDARA$(84:84) = "1"            ! Subfile control display on
00750   INDARA$(85:85) = "1"            ! Subfile display on
00760   GOSUB GENGRAPH                  ! Draw the chart
00770   WRITE #5,EXTDESCR "SFCTL", INDIC INDARA$:
00780   WRITE #5, EXTDESCR "OVERL", INDIC INDARA$:
00790   READ  #5,EXTDESCR "SFCTL", INDIC INDARA$:
00800   !************************************************************
00810   !* Test the input for the next action to be taken:
00820   !*
00830   !*  CF1  -  Exit the application (indicator 98)
00840   !*  ENTER - Update the data or add the record
00850   !*          to the file or to the display
00860   !************************************************************
00870   IF INDARA$(98:98) = "1" THEN GOSUB TERM  ELSE GOSUB UPDATE
00880   !************************************************************
00890   !* Return to redisplay the subfile with any changed data.
00900   !************************************************************
00910 GOTO REDISPLAY
00920 !************************************************************
00930 !*
00940 !* SUBROUTINE TO TERMINATE GRAPHICS
00950 !*
00960 !************************************************************
00970 TERM:                             ! Terminate graphics
00980   CALL GDDM('FSTERM')
00990 STOP
01000 RETURN
01010 !************************************************************
01020 !*
01030 !* SUBROUTINE TO ADD A RECORD TO THE SUBFILE AND ADD THE
01040 !* THE BEGINNING AND ENDING DATA TO THE PLOT ARRAYS
01050 !*
01060 !************************************************************
01070 ADDSUBF: READ #6,KEY=PROJ$,EXTDESCR "SCHED": NOKEY 1120
01080   READ  #7,KEY=PROJ$,EXTDESCR "ACCT":   NOKEY 1120
01090   REWRITE #5, EXTDESCR "SFLA", INDIC INDARA$, REC=I:
01100   STARTARRAY(I) = VAL(BEGDAT$(1:2))
01110   ENDARRAY(I) = VAL(ENDDAT$(1:2))
01120 RETURN
01130 !************************************************************
01140 !*
01150 !* SUBROUTINE TO READ THE SUBFILE FOR ANY UPDATES
01160 !* OR ADDITIONS TO THE SCHEDULE AND ACCOUNTING FILES
01170 !*
```

```
01180 !************************************************************
01190 UPDATE:                              ! Update changed records
01200   FOR I = 1 TO NUM
01210     READ #5, EXTDESCR "SFLA",REC=I:
01220     BDATE$ = BEGDAT$: EDATE$ = ENDDAT$
01230     TCOST = COST: TREV = REV
01240     READ #6, KEY=PROJ$,EXTDESCR "SCHED": NOKEY 1300
01250     BEGDAT$ = BDATE$: ENDDAT$ = EDATE$
01260     REWRITE #6,EXTDESCR "SCHED":
01270     READ #7, KEY=PROJ$,EXTDESCR "ACCT": NOKEY 1300
01280     COST = TCOST: REV = TREV
01290     REWRITE #7,EXTDESCR "ACCT":
01300   NEXT I
01310   IF NUM < 6 THEN GOSUB READADD
01320 RETURN
01330 !************************************************************
01340 !*
01350 !* SUBROUTINE TO READ ANY ADDED PROJECTS IN THE SUBFILE AND
01360 !* ADD THEM TO THE DATABASE FILE IF NOT ALREADY THERE
01370 !*
01380 !************************************************************
01390 READADD:                             ! Add records to files and/or
01400   FOR I = NUM+1 TO 6                 ! to the project list
01410     READ #5, EXTDESCR "SFLA",REC=I:
01420     IF PROJ$ = '      ' THEN GOTO 1460
01430     WRITE #6,EXTDESCR "SCHED":  DUPKEY 1450
01440     WRITE #7,EXTDESCR "ACCT":
01450     GOSUB ADDLIST
01460   NEXT I
01470 RETURN
01480 !************************************************************
01490 !*
01500 !* SUBROUTINE TO ADD PROJECT NUMBER TO THE PROJECT NUMBER LIST
01510 !*
01520 !************************************************************
01530 ADDLIST:                             ! Add to project list
01540   PROJLIST$(I) = PROJ$
01550   NUM = NUM + 1                      ! Increment project number
01560 RETURN
01570 !************************************************************
01580 !*
01590 !* SUBROUTINE TO GENERATE THE CHART OF PROJECT SCHEDULES
01600 !*
01610 !************************************************************
01620 GENGRAPH: !
01630   FOR I = 1 TO NUM                   ! Fill the data array-merge
01640       DATAARRAY(I) = STARTARRAY(I)   ! Starting and ending arrays
01650   NEXT I
01660   FOR I = NUM+1 TO 2*NUM
01670     DATAARRAY(I) = ENDARRAY(I-NUM)
01680   NEXT I
01690   !  Construct x-axis labels from the project list
01700   XLABEL$=''                         ! Initialize the labels
01710   FOR I = 1 TO NUM
01720    XLABEL$ = XLABEL$&PROJLIST$(I)
01730   NEXT I
01740   CALL GDDM('CHXLAB',NUM,6,XLABEL$) ! Set the x-axis labels
01750   CALL GDDM('CHBAR',2,NUM,DATAARRAY())    ! Draw the bar chart
01760   CALL GDDM('CHDRAX')                ! Draw the axis, grid lines
01770   CALL GDDM('FSFRCE')                ! Force the chart to display
```

```
01780   CALL GDDM('GSCLR')                ! Clear graphics
01790   CALL GDDM('CHSTRT')               ! Reset processing state
01800 RETURN
01810 !*************************************************************
01820 !*
01830 !* SUBROUTINE TO SET THE CHART ATTRIBUTES
01840 !*
01850 !*************************************************************
01860 SETGRAPH: !
01870   CALL GDDM('FSINIT')               ! Initialize graphics
01880   CALL GDDM('GSFLD',1,1,24,80)      ! Set the graphics field
01890   XPS=1.0: YPS=0.7                  ! Picture space variables
01900   CALL GDDM('GSPS',XPS,YPS)         ! Set the picture space ratio
01910   X1=0.0: X2=1.0: Y1=0.3: Y2=0.7    ! Chart area values
01920   CALL GDDM('CHAREA',X1,X2,Y1,Y2)   ! Set the chart area
01930   AXSARRAY(4)=1                     ! Axis attribute array
01940   CALL GDDM('CHAATT',6,AXSARRAY())  ! Set axis attributes
01950   TTLARRAY(1)=1: TTLARRAY(2)=2      ! Axis title array
01960   CALL GDDM('CHTATT',2,TTLARRAY())  ! Set chart title attributes
01970   LBLARRAY(1)=4: LBLARRAY(2)=2      ! Axis label array
01980   LBLARRAY(4)=200                   ! Axis label array
01990   CALL GDDM('CHLATT',4,LBLARRAY())  ! Set chart label attributes
02000   GRDARRAY(3)=0: GRDARRAY(4)=1      ! Grid attribute array
02010   GRDARRAY(5)=0: GRDARRAY(6)=0      ! Grid attribute array
02020   CALL GDDM('CHGATT',6,GRDARRAY())  ! Set grid attributes
02030   HDGARRAY(1)=2: HDGARRAY(2)=2      ! Heading attribute array
02040   CALL GDDM('CHHATT',2,HDGARRAY())  ! Set heading attributes
02050   CALL GDDM('CHHEAD',23,'YEARLY PROJECT SCHEDULE')! Chart heading
02060   COLARRAY(1)=8: COLARRAY(2)=2      ! Component color array
02070   CALL GDDM('CHCOL',2,COLARRAY())   ! Set colors for the chart
02080   PATARRAY(1)=16                    ! Component pattern array
02090   CALL GDDM('CHPAT',1,PATARRAY())   ! Set pattern for the chart
02100   GAPRATIO=3.0                      ! Bar gap ratio value
02110   CALL GDDM('CHGAP',GAPRATIO)       ! Set gap-to-bar ratio to 3:1
02120   CALL GDDM('CHSET','XVER')         ! Use horizontal orientation
02130   CALL GDDM('CHSET','CBAR')         ! Use composite bar chart
02140   CALL GDDM('CHVMAR',7,1)           ! Set vertical margins
02150   CALL GDDM('CHHMAR',3,3)           ! Set horizontal margins
02160   CALL GDDM('CHXTTL',7,'PROJECT')   ! x-axis title
02170   CALL GDDM('CHYTTL',5,'MONTH')     ! y-axis title
02180   CALL GDDM('CHXSET','PLAIN')       ! Suppress x-axis tick marks
02190   CALL GDDM('CHXSET','ATABOVE')     ! Put x-axis title above axis
02200   CALL GDDM('CHYMTH',1)             ! Use months for y-axis labels
02210   YRNG1=1.0: YRNG2=12.0             ! y-range values
02220   CALL GDDM('CHYRNG',YRNG1,YRNG2)   ! Set y-axis range to 12 months
02230   CALL GDDM('CHSET','LETT')         ! Abbreviate month to 1 letter
02240   CALL GDDM('CHYSET','GRID')        ! Use grid lines from y axis
02250   CALL GDDM('CHSET','NDRAW')        ! Wait to draw axes and grid
02260 RETURN
02270 END
```

COBOL/400 Multiple-Pie Chart Program



This COBOL/400 program draws two pie charts. The first pie uses absolute data, the second relative data (the sum of the chart data value array elements for the second chart must equal 100 for a complete pie).

```
-A+++B++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CPGFAPP1.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  IBM-S38.
 OBJECT-COMPUTER.  IBM-S38.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*****************************************************************
*   Parameters used in graphics routines
*****************************************************************
 77  FKTYPE PIC S9(5) COMP-4.
 77  FKNUM PIC S9(5) COMP-4.
 77  NUM   PIC S9(5) COMP-4.
 77  STRING-LENGTH PIC S9(5) COMP-4.
 77  X PIC S9(4)V9 COMP-3.
 77  Y PIC S9(4)V9 COMP-3.
 77  COMPNUMBER PIC S9(5) COMP-4.
 77  XINT1 PIC S9(4)V9 COMP-3.
 77  XINT2 PIC S9(4)V9 COMP-3.
 77  YINT1 PIC S9(4)V9 COMP-3.
 77  YINT2 PIC S9(4)V9 COMP-3.
 77  ROW1 PIC S9(5) COMP-4.
 77  ROW2 PIC S9(5) COMP-4.
 77  COL1 PIC S9(5) COMP-4.
```

```
 77  COL2 PIC S9(5) COMP-4.
 77  LMARG PIC S9(5) COMP-4.
 77  RMARG PIC S9(5) COMP-4.
 77  OFFSET1 PIC 99V9 COMP-3.
 77  OFFSET2 PIC 99V9 COMP-3.
 77  SEGNUMBER PIC S9(5) COMP-4.
 77  CHARTNUM PIC S9(5) COMP-4.
 77  COLOR PIC S9(5) COMP-4.
 77  POSCDE PIC X(2).
 77  STRNG PIC X(50).
 77  KEYP1 PIC X(1).
 77  KEYP2 PIC X(1).
 77  KEYP3 PIC X(1).
 77  CHSETVALUE PIC X(8).
*****************************************************************
*   Presentation Graphics and GDDM routines used in the program
*****************************************************************
 77  FSINIT PIC X(8) VALUE "FSINIT  ".
 77  GSFLD  PIC X(8) VALUE "GSFLD   ".
 77  GSPS   PIC X(8) VALUE "GSPS    ".
 77  GSSEG  PIC X(8) VALUE "GSSEG   ".
 77  GSSCLS PIC X(8) VALUE "GSSCLS  ".
 77  GSCOL  PIC X(8) VALUE "GSCOL   ".
 77  GSMOVE PIC X(8) VALUE "GSMOVE  ".
 77  GSCHAP PIC X(8) VALUE "GSCHAP  ".
 77  CHAREA PIC X(8) VALUE "CHAREA  ".
 77  CHPAT  PIC X(8) VALUE "CHPAT   ".
 77  CHCOL  PIC X(8) VALUE "CHCOL   ".
 77  CHVMAR PIC X(8) VALUE "CHVMAR  ".
 77  CHSET  PIC X(8) VALUE "CHSET   ".
 77  CHKATT PIC X(8) VALUE "CHKATT  ".
 77  CHKEY  PIC X(8) VALUE "CHKEY   ".
 77  CHKEYP PIC X(8) VALUE "CHKEYP  ".
 77  CHNATT PIC X(8) VALUE "CHNATT  ".
 77  CHLATT PIC X(8) VALUE "CHLATT  ".
 77  CHVATT PIC X(8) VALUE "CHVATT  ".
 77  CHNOTE PIC X(8) VALUE "CHNOTE  ".
 77  CHNOFF PIC X(8) VALUE "CHNOFF  ".
 77  CHPIE  PIC X(8) VALUE "CHPIE   ".
 77  ASREAD PIC X(8) VALUE "ASREAD  ".
 77  FSTERM PIC X(8) VALUE "FSTERM  ".
 77  CHRNIT PIC X(8) VALUE "CHRNIT  ".
*****************************************************************
*   Initialize the data arrays for the two pie charts
*   and the arrays for color and pattern attributes
*****************************************************************
 01  DATA-VARIABLES.
     03 ARRY1.
        05 PIE1 OCCURS 10 TIMES PIC 9999V9 COMP-3.
     03 ARRY2.
        05 PIE2 OCCURS 10 TIMES PIC 9999V9 COMP-3.
     03 ARRY3.
        05 COL4ARRY OCCURS 10 TIMES PIC S9(5) COMP-4.
     03 ARRY4.
        05 PATARRAY OCCURS 10 TIMES PIC S9(5) COMP-4.
     03 ARRY5.
        05 KATTS OCCURS 10 TIMES PIC S9(5) COMP-4.
     03 ARRY6.
        05 LATTS OCCURS 10 TIMES PIC S9(5) COMP-4.
     03 ARRY6.
```

```
           05 VATTS OCCURS 10 TIMES PIC S9(5) COMP-4.
        03 ARRY8.
           05 NATTS OCCURS 10 TIMES PIC S9(5) COMP-4.
 PROCEDURE DIVISION.
 GRAPHICS.
*************************************************************
*  Initialize graphics
*************************************************************
     CALL "GDDM" USING FSINIT.
*************************************************************
*  Specify data values for the two pies
*************************************************************
     MOVE 110.0 TO PIE1(1).
     MOVE 90.0 TO PIE1(2).
     MOVE 85.0 TO PIE1(3).
     MOVE 75.0 TO PIE1(4).
     MOVE 40.0 TO PIE2(1).
     MOVE 20.0 TO PIE2(2).
     MOVE 15.0 TO PIE2(3).
     MOVE 25.0 TO PIE2(4).
*************************************************************
*  Specify the color attributes for the 4 pie slices
*************************************************************
     MOVE 2 TO COL4ARRY(1).
     MOVE 3 TO COL4ARRY(2).
     MOVE 4 TO COL4ARRY(3).
     MOVE 5 TO COL4ARRY(4).
*************************************************************
*  Set the graphics field and the graphics picture space
*************************************************************
     MOVE 1 TO ROW1.
     MOVE 1 TO COL1.
     MOVE 24 TO ROW2.
     MOVE 80 TO COL2.
     CALL "GDDM" USING GSFLD, ROW1, COL1, ROW2, COL2.
     MOVE 1.0 TO X.
     MOVE 0.6 TO Y.
     CALL "GDDM" USING GSPS, X, Y.
*************************************************************
*  Create segment 1 and define the chart headings
*************************************************************
     MOVE 1 TO SEGNUMBER.
     CALL "GDDM" USING GSSEG, SEGNUMBER.
     MOVE 6 TO COLOR.
     CALL "GDDM" USING GSCOL, COLOR.
     MOVE 28 TO X.
     MOVE 95 TO Y.
     CALL "GDDM" USING GSMOVE, X, Y.
     MOVE 27 TO STRING-LENGTH.
     MOVE "HOUSEHOLD EXPENSE BREAKDOWN" TO STRNG.
     CALL "GDDM" USING GSCHAP, STRING-LENGTH, STRNG.
     MOVE 10.0 TO X.
     MOVE 85.0 TO Y.
     CALL "GDDM" USING GSMOVE, X, Y.
     MOVE 15 TO STRING-LENGTH.
     MOVE "OVERALL EXPENSE           " TO STRNG.
     CALL "GDDM" USING GSCHAP, STRING-LENGTH, STRNG.
     MOVE 60.0 TO X.
     MOVE 85.0 TO Y.
     CALL "GDDM" USING GSMOVE, X, Y.
```

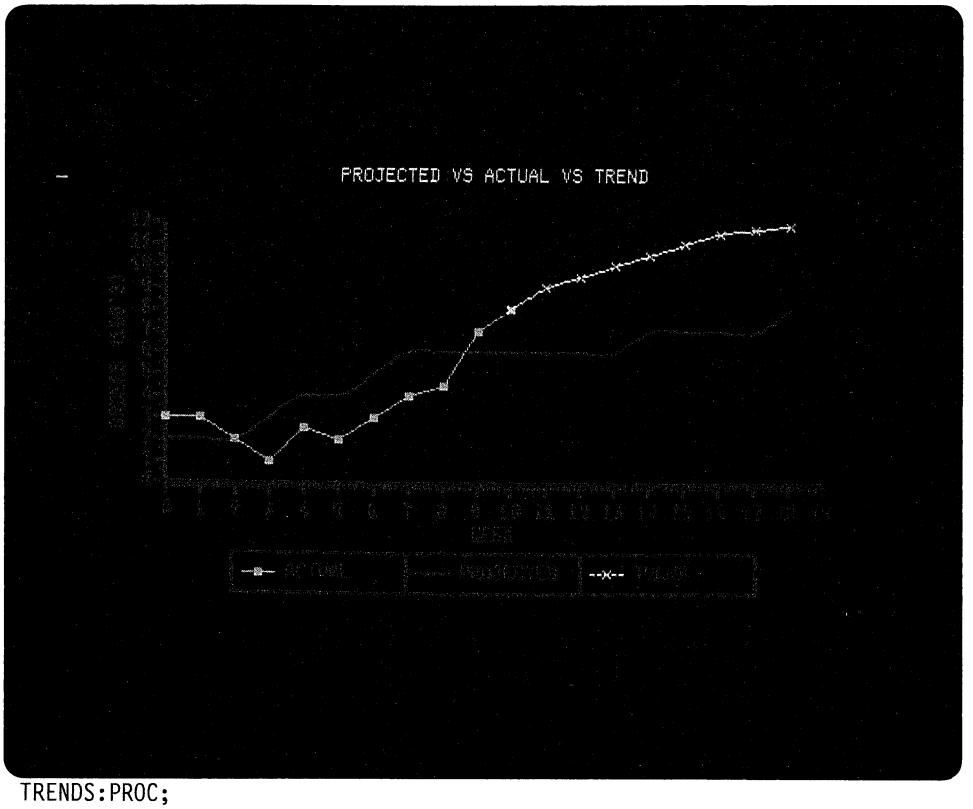
```
     MOVE 21 TO STRING-LENGTH.
     MOVE "MISCELLANEOUS EXPENSE      " TO STRNG.
     CALL "GDDM" USING GSCHAP, STRING-LENGTH, STRNG.
*************************************************************
*  Define the chart area for the first pie
*************************************************************
     MOVE 0.0 TO XINT1.
     MOVE 0.6 TO XINT2.
     MOVE 0.0 TO YINT1.
     MOVE 0.6 TO YINT2.
     CALL "GDDM" USING CHAREA, XINT1, XINT2, YINT1, YINT2.
*************************************************************
*  Define the component patterns and colors for the first pie
*************************************************************
     MOVE 2 TO PATARRAY(1).
     MOVE 4 TO PATARRAY(2).
     MOVE 6 TO PATARRAY(3).
     MOVE 8 TO PATARRAY(4).
     MOVE 4 TO COMPNUMBER.
     CALL "GDDM" USING CHPAT, COMPNUMBER, PATARRAY.
     CALL "GDDM" USING CHCOL, COMPNUMBER, COL4ARRY.
*************************************************************
*  Specify margins, absolute values, and value text
*************************************************************
     MOVE 5 TO RMARG.
     MOVE 1 TO LMARG.
     CALL "GDDM" USING CHVMAR, LMARG, RMARG.
     MOVE "ABPIE " TO CHSETVALUE.
     CALL "GDDM" USING CHSET, CHSETVALUE.
     MOVE "PIEKEY" TO CHSETVALUE.
     CALL "GDDM" USING CHSET, CHSETVALUE.
     MOVE 1 TO VATTS(1).
     MOVE 2 TO VATTS(2).
     MOVE 0 TO VATTS(3).
     MOVE 150 TO VATTS(4).
     CALL "GDDM" USING CHVATT, COMPNUMBER, VATTS.
     MOVE "VALUES" TO CHSETVALUE.
     CALL "GDDM" USING CHSET, CHSETVALUE.
*************************************************************
*  Position the legend, define the key labels for the first pie
*************************************************************
     MOVE "H" TO KEYP1.
     MOVE "B" TO KEYP2.
     MOVE "C" TO KEYP3.
     CALL "GDDM" USING CHKEYP, KEYP1, KEYP2, KEYP3.
     MOVE 7 TO KATTS(1).
     MOVE 2 TO KATTS(2).
     MOVE 0 TO KATTS(3).
     MOVE 150 TO KATTS(4).
     CALL "GDDM" USING CHKATT, COMPNUMBER, KATTS.
     MOVE 7 TO STRING-LENGTH.
     MOVE "RENT   UTILITYTRANPRTMISCELL." TO STRNG.
     CALL "GDDM" USING CHKEY, COMPNUMBER, STRING-LENGTH, STRNG.
     CALL "GDDM" USING GSSCLS.
*************************************************************
*  Draw the first pie chart and re-initialize to define second
*************************************************************
     MOVE 1 TO CHARTNUM.
     CALL "GDDM" USING CHPIE, CHARTNUM, COMPNUMBER, PIE1.
```

```
     CALL "GDDM" USING CHRNIT.
*************************************************************
*  Define the chart area for the second pie
*************************************************************
     MOVE 0.6 TO XINT1.
     MOVE 1.0 TO XINT2.
     MOVE 0.0 TO YINT1.
     MOVE 0.6 TO YINT2.
     CALL "GDDM" USING CHAREA, XINT1, XINT2, YINT1, YINT2.
*************************************************************
*  Specify margins
*************************************************************
     MOVE 1 TO RMARG.
     MOVE 1 TO LMARG.
     CALL "GDDM" USING CHVMAR, LMARG, RMARG.
*************************************************************
*  Define the patterns for the second pie
*************************************************************
     MOVE 3 TO PATARRAY(1).
     MOVE 6 TO PATARRAY(2).
     MOVE 9 TO PATARRAY(3).
     MOVE 12 TO PATARRAY(4).
     CALL "GDDM" USING CHPAT, COMPNUMBER, PATARRAY.
*************************************************************
*  Specify spider labels and percentage values
*************************************************************
     MOVE 250 TO KATTS(4).
     CALL "GDDM" USING CHKATT, COMPNUMBER, KATTS.
     MOVE "SPIDER" TO CHSETVALUE.
     CALL "GDDM" USING CHSET, CHSETVALUE.
     MOVE "PERPIE" TO CHSETVALUE.
     CALL "GDDM" USING CHSET, CHSETVALUE.
*************************************************************
*  Specify labels for the second pie
*************************************************************
     MOVE 9 TO STRING-LENGTH.
     MOVE "ENTRTNMNTMEDICAL  CLOTHES  HOBBIES  " TO STRNG.
     CALL "GDDM" USING CHKEY, COMPNUMBER, STRING-LENGTH, STRNG.
*************************************************************
*  Draw the second pie and write a chart note
*************************************************************
     CALL "GDDM" USING CHPIE, CHARTNUM, COMPNUMBER, PIE2.
     MOVE 0 TO OFFSET1.
     MOVE 5 TO OFFSET2.
     MOVE 2 TO NATTS(1).
     MOVE 2 TO NATTS(2).
     MOVE 0 TO NATTS(3).
     MOVE 250 TO NATTS(4).
     CALL "GDDM" USING CHNATT, COMPNUMBER, NATTS.
     CALL "GDDM" USING CHNOFF, OFFSET1, OFFSET2.
     MOVE "C1" TO POSCDE.
     MOVE 33 TO STRING-LENGTH.
     MOVE "MISC. EXPENSES ARE;21% OF TOTAL" TO STRNG.
     CALL "GDDM" USING CHNOTE, POSCDE, STRING-LENGTH, STRNG.
*************************************************************
*  Send the pies to the screen and terminate graphics
*************************************************************
     CALL "GDDM" USING ASREAD, FKTYPE, FKNUM, NUM.
     CALL "GDDM" USING FSTERM.
     STOP RUN.
```

```
END-GRAPHICS.
```

PL/I Planned Versus Actual Versus Trend Program

This program uses data from arrays within the program to plot three different chart components. Each time a component is plotted, one part of the legend is constructed. After each component is drawn, the CHRNIT routine reinitializes Presentation Graphics and the chart is again defined.



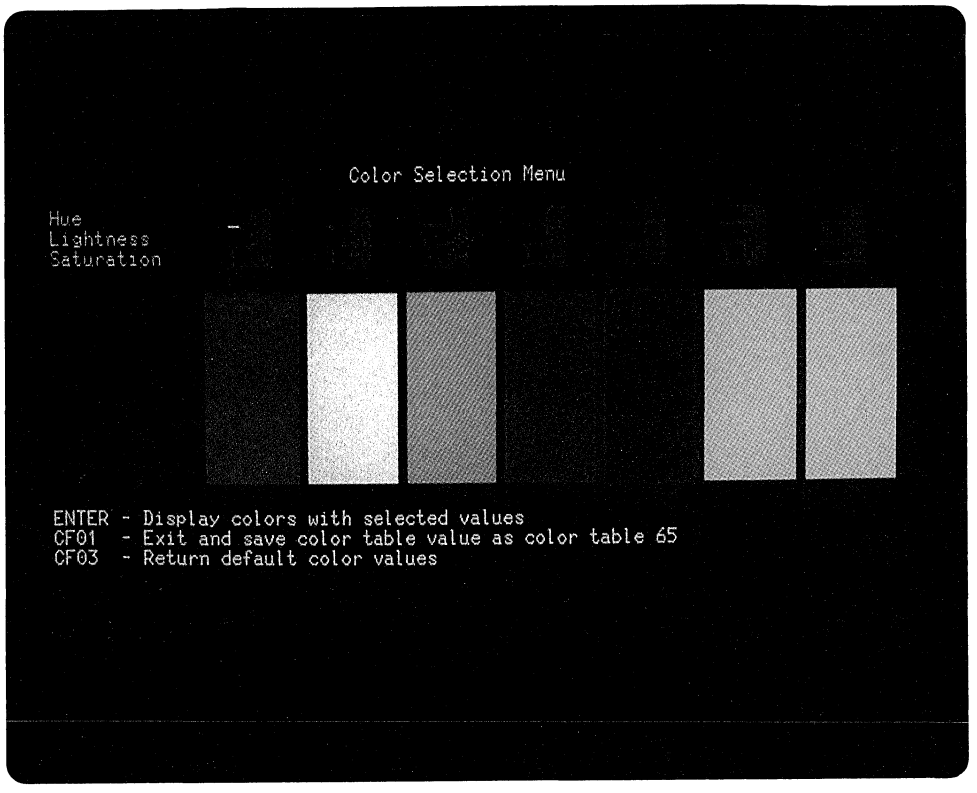

```
TRENDS:PROC;
/*********************************************************************/
/*        PROJECTED ORDERS VERSUS ACTUAL VERSUS TREND                */
/*********************************************************************/
%INCLUDE (ADMUPINA);
%INCLUDE (ADMUPINC);                /*  INCLUDES   */
%INCLUDE (ADMUPINF);
%INCLUDE (ADMUPING);

DCL
  YELLOW(1) FIXED BIN(31) STATIC INIT(6);
DCL
  AXIS_ATTR(6) FIXED BIN(31) STATIC INIT(1,7,1,1,7,1);
DCL
  MARKERS(3) FIXED BIN(31) STATIC INIT(8,7,6);
DCL
  (ATTENTION_TYPE,
   ATTENTION_MOD,
   FLD_COUNT) FIXED BIN(31) STATIC INIT(0);

/*********************************************************************/
/*                DATA ARRAYS FOR CHART COMPONENTS                   */
/*********************************************************************/
DCL
  Y_ACTUAL(11) FLOAT DEC(6) STATIC INIT(3,3,2,1,2.5,2,3,4,4.5,7,8);
```

```
DCL
  Y_PROJECTED(19) FLOAT DEC(6) STATIC INIT(
       2,2,2,3,4,4,5,6,6,6,6,6,6,6,7,7,7,7,8);
DCL
  Y_TREND(10) FLOAT DEC(6) STATIC
    INIT(8,9,9.5,10,10.5,11,11.5,11.7,11.9,12);
DCL
  X_ACTUAL(11) FLOAT DEC(6) STATIC INIT(0,1,2,3,4,5,6,7,8,9,10);
DCL
  X_PROJECTED(20) FLOAT DEC(6) STATIC INIT
      (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19);
DCL
  X_TREND(10) FLOAT DEC(6) STATIC INIT
       (10,11,12,13,14,15,16,17,18,19);

/*********************************************************************/
/*                      CHART CONSTRUCTION                           */
/*********************************************************************/
  CALL FSINIT;                      /* Initialize graphics           */
  CALL CHHATT(1,YELLOW);            /* Set heading attributes        */
  CALL CHHEAD(28,'PROJECTED VS ACTUAL VS TREND');  /* Write heading */
  CALL CHMARK(3,MARKERS);           /* Use markers from defined tbl  */
/*********************************************************************/
/*         Define chart and plot first component (ACTUAL)            */
/*********************************************************************/
  CALL CHHMAR(8,3);                 /* Set bot margin 8 rows, top 3  */
  CALL CHAATT(6,AXIS_ATTR);         /* Set axis attributes           */
  CALL CHXTTL( 4,'WEEK');           /* Write 4-char x-axis title     */
  CALL CHYTTL(14,'ORDERS (100''S)');  /* 14-character y-axis title  */
  CALL CHXRNG(0,19);                /* Set x-axis range to 0 - 19    */
  CALL CHYRNG(0,12);                /* Set y-axis range to 0 - 12    */
  CALL CHXTIC(1,5);                 /* x = 5 minor ticks between maj */
  CALL CHYTIC(1,1);                 /* y = 1 minor tick between maj  */
  CALL CHCOL(1,2);                  /* Component color red           */
  CALL CHLT(1,0);                   /* Component line type solid     */
  CALL CHKEYP('H','B','C');         /* Horizontal legend, bot, center*/
  CALL CHKOFF(-16.0,2.5);           /* Position legend box           */
  CALL CHSET('KBOX');               /* Enclose legend in box         */
  CALL CHKEY(1,10,'ACTUAL    ');   /* Write first legend key label  */
  CALL CHPLOT(1,11,X_ACTUAL,Y_ACTUAL);   /* Plot first component    */
/*********************************************************************/
/*          Reinitialize Presentation Graphics,                      */
/*        redefine chart and plot second component                   */
/*                      (PROJECTED)                                  */
/*********************************************************************/
  CALL CHRNIT;                      /* Reinitialize Presentation G   */
  CALL CHAATT(6,AXIS_ATTR);         /* Set axis attributes           */
  CALL CHHMAR(8,3);                 /* Set bot margin 8 rows, top 3  */
  CALL CHXRNG(0,19);                /* Set x-axis range to 0 - 19    */
  CALL CHYRNG(0,12);                /* Set y-axis range to 0 - 12    */
  CALL CHXSET('NOAXIS');            /* Suppress:   x axis            */
  CALL CHXSET('NOLAB');             /*             x-axis labels     */
  CALL CHXSET('PLAIN');             /*             x-axis tick marks */
  CALL CHYSET('NOAXIS');            /*             y axis            */
  CALL CHYSET('NOLAB');             /*             y-axis labels     */
  CALL CHYSET('PLAIN');             /*             y-axis tick marks */
  CALL CHSET('NOMARKERS');          /*             component markers */
  CALL CHSET('CURVE');              /* Use curved line for component */
  CALL CHFINE(15);                  /* Curve value 15                */
  CALL CHKOFF(0,2.5);               /* Position legend box           */
```

```
  CALL CHSET('KBOX');                 /* Enclose legend in box         */
  CALL CHKEYP('H','B','C');           /* Horizontal legend, bot, center*/
  CALL CHKEY(1,10,'PROJECTED ');      /* Write second legend key label */
  CALL CHCOL(1,5);                    /* Component color turquoise     */
  CALL CHLT(1,1);                     /* Component line type dotted    */
  CALL CHPLOT(1,19,X_PROJECTED,Y_PROJECTED);  /* Plot second compnt */
/*********************************************************************/
/*          Reinitialize Presentation Graphics,                      */
/*        redefine chart and plot third component                    */
/*                         (TREND)                                   */
/*********************************************************************/
  CALL CHRNIT;                        /* Reinitialize Presentation G   */
  CALL CHAATT(6,AXIS_ATTR);           /* Set axis attributes           */
  CALL CHHMAR(8,3);                   /* Set bot margin 8 rows, top 3  */
  CALL CHXRNG(0,19);                  /* Set x-axis range to 0 - 19    */
  CALL CHYRNG(0,12);                  /* Set y-axis range to 0 - 12    */
  CALL CHXSET('NOAXIS');              /* Suppress:   x axis            */
  CALL CHXSET('NOLAB');               /*             x-axis labels     */
  CALL CHXSET('PLAIN');               /*             x-axis tick marks */
  CALL CHYSET('NOAXIS');              /*             y axis            */
  CALL CHYSET('NOLAB');               /*             y-axis labels     */
  CALL CHYSET('PLAIN');               /*             y-axis tick marks */
  CALL CHSET('CURVE');                /* Use curved line for component */
  CALL CHFINE(15);                    /* Curve value 15                */
  CALL CHKOFF(16.0,2.5);              /* Position legend box           */
  CALL CHSET('KBOX');                 /* Enclose legend in box         */
  CALL CHKEYP('H','B','C');           /* Horizontal legend, bot, center*/
  CALL CHKEY(1,10,'TREND     ');      /* Write third legend key label  */
  CALL CHCOL(1,6);                    /* Component color yellow        */
  CALL CHLT(1,2);                     /* Component line type dashed    */
  CALL CHPLOT(1,9,X_TREND,Y_TREND);/* Plot third component             */
/*********************************************************************/
/*             Write note, send chart to display,                    */
/*            and terminate Presentation Graphics                    */
/*********************************************************************/
  CALL CHNOTE('BL',26,'PRESS ENTER TO END PROGRAM');  /* Write note */
  CALL ASREAD(ATTENTION_TYPE,ATTENTION_MOD,FLD_COUNT); /* Send      */
  CALL FSTERM;                                         /* Terminate */
END TRENDS;
```

PL/I GDDM Color Table Application

This program is useful for setting the color table for your program. If you add a routine to your graphics program that selects color table entry 65 ('GSCT',65), you can call this program from any of your graphics programs and set the color table values used to display the picture *interactively*.



This is the display file (named COLORS) used with the graphics program:

```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     A                                      DSPSIZ(24 80 *DS3)
     A                                      INDARA
     A          R COLORS
     A                                      CF01(01 'end the program')
     A                                      CF02(02 'return default color -
     A                                      values')
     A                                      ALWGPH
     A                                      OVERLAY
     A                                    1 29'Color Selection Menu'
     A                                      COLOR(YLW)
     A                                    3  2'Hue       '
     A                                      COLOR(RED)
     A            FLD065          3Y 2B  3 18EDTWRD('0.  ')
     A            FLD066          3Y 2B  3 27EDTWRD('0.  ')
     A            FLD067          3Y 2B  3 36EDTWRD('0.  ')
     A            FLD068          3Y 2B  3 45EDTWRD('0.  ')
     A            FLD069          3Y 2B  3 54EDTWRD('0.  ')
     A            FLD070          3Y 2B  3 63EDTWRD('0.  ')
     A            FLD071          3Y 2B  3 72EDTWRD('0.  ')
     A                                    4  2'Lightness '
     A                                      COLOR(RED)
     A            FLD072          3Y 2B  4 18EDTWRD('0.  ')
     A            FLD073          3Y 2B  4 27EDTWRD('0.  ')
```

```
A            FLD074         3Y 2B  4 36EDTWRD('0.  ')
A            FLD075         3Y 2B  4 45EDTWRD('0.  ')
A            FLD076         3Y 2B  4 54EDTWRD('0.  ')
A            FLD077         3Y 2B  4 63EDTWRD('0.  ')
A            FLD078         3Y 2B  4 72EDTWRD('0.  ')
A                                    5  2'Saturation'
A                                       COLOR(RED)
A            FLD079         3Y 2B  5 18EDTWRD('0.  ')
A            FLD080         3Y 2B  5 27EDTWRD('0.  ')
A            FLD081         3Y 2B  5 36EDTWRD('0.  ')
A            FLD082         3Y 2B  5 45EDTWRD('0.  ')
A            FLD083         3Y 2B  5 54EDTWRD('0.  ')
A            FLD084         3Y 2B  5 63EDTWRD('0.  ')
A            FLD085         3Y 2B  5 72EDTWRD('0.  ')
A                                   18  2'ENTER - Display colors with s-
A                                       elected values'
A                                       COLOR(WHT)
A                                   19  2'CF01  - Exit and save color t-
A                                       able value as color table 65'
A                                       COLOR(WHT)
A                                   20  2'CF02  - Return default color +
A                                       values'
A                                       COLOR(WHT)
```

This is the PL/I program you can call from your programs to define the color table you select:

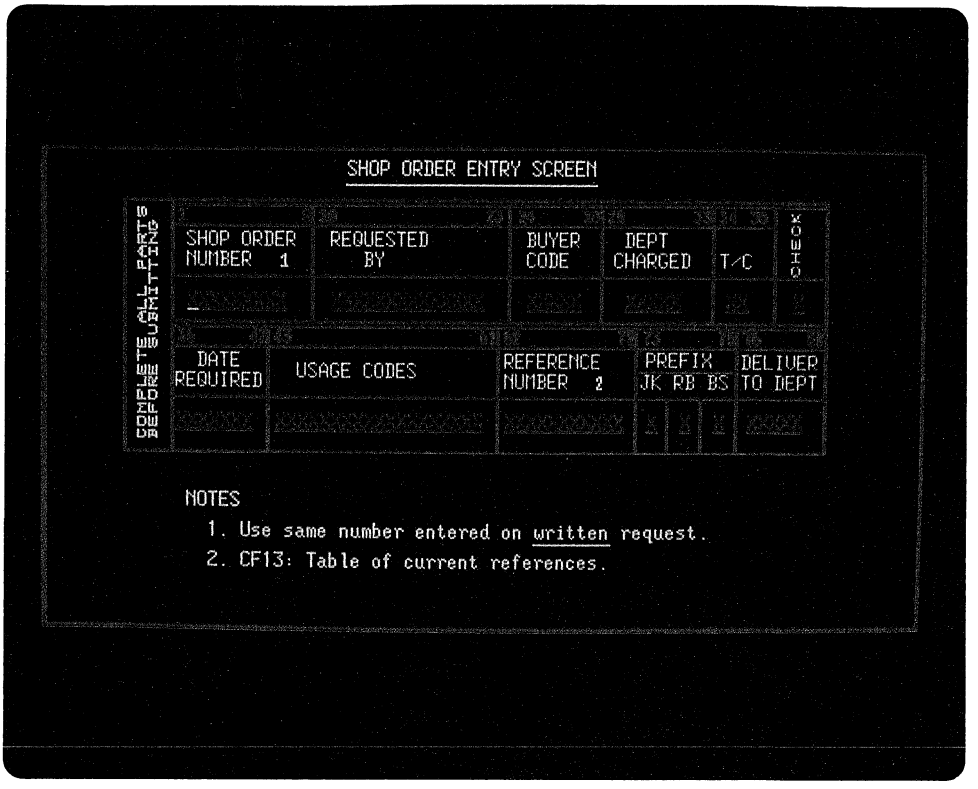
```
COLOR:   PROC;
/*********************************************************************/
/*                                                                   */
/*  FUNCTION: Program to interactively generate a color table        */
/*                                                                   */
/*  INPUT:    None                                                   */
/*                                                                   */
/*  OUTPUT:   Color table #65 is defined;  it must be selected by    */
/*            the 'calling' program using GDDM routine GSCT.         */
/*                                                                   */
/*  ASSUMPTIONS:  1) The current device supports the color table.    */
/*                2) Page identifier 99 is not in use.               */
/*                3) The current graphics screen contents are        */
/*                   expendable.                                     */
/*                4) The file 'COLORS' exists in a library on the    */
/*                   library list.                                   */
/*                5) The page in use when this program is called     */
/*                   must be reselected when this program returns    */
/*                   control to the calling program.                 */
/*                                                                   */
/*********************************************************************/
DCL
  CFILE FILE RECORD ENV(INTERACTIVE) UPDATE;
/*********************************************************************/
/* General variable declarations                                     */
/*********************************************************************/
DCL
  COLOR_INDEX FIXED BIN(31),     /* Color index                      */
  I           FIXED BIN(31),     /* Temporary loop index             */
  K           FIXED BIN(31),     /* Temporary loop index             */
  J           FIXED BIN(31),     /* Temporary loop index             */
  START_X     FLOAT DEC(6),      /* Starting x coord for polygons    */
  END_X       FLOAT DEC(6),      /* Ending x coord for polygons      */
  SEG_ID FIXED BIN(31) STATIC INIT(99),
```

```
                                  /* Segment identifier            */
  INCR_X FLOAT DEC(6) STATIC INIT(9);
                                  /* x coord increment for polygons*/
 DCL 1 DSPREC AUTOMATIC,
 %INCLUDE COLORS(COLORS,RECORD); /* DSPF record format            */
 DCL 1 HLS BASED(RECPTR),
      2 D_HUE(7) PIC '9V9R',
      2 D_LIGHT(7) PIC '9V9R',
      2 D_SATUR(7) PIC '9V9R';
 DCL RECPTR PTR ;
 DCL CFKEYS   CHAR(3);            /* Aid key indicators            */
%INCLUDE SYSLIB(ADMUPLNB);        /* GDDM entry module includes    */

/*****************************************************************/
/* Declare the color table variables                             */
/*****************************************************************/
 DCL HUE(7)     FLOAT DEC (6)
                INIT(0,.33333,.16666,.66666,.83333,.5,0) STATIC,
     LIGHT(7)   FLOAT DEC (6)
                INIT(.5,.5,.5,.5,.5,.5,1) STATIC,
     SATUR(7)   FLOAT DEC (6)
                INIT(1,1,1,1,1,1,0) STATIC;

/*****************************************************************/
/* Declare the default color table values                        */
/*****************************************************************/
 DCL S_HUE(7)   FLOAT DEC (6)
                INIT(0,.33333,.16666,.66666,.83333,.5,0) STATIC,
     S_LIGHT(7) FLOAT DEC (6)
                INIT(.5,.5,.5,.5,.5,.5,1) STATIC,
     S_SATUR(7) FLOAT DEC (6)
                INIT(1,1,1,1,1,1,0) STATIC;
/*****************************************************************/
/* Initialize the display file fields                            */
/*****************************************************************/
RECPTR = ADDR(DSPREC);
DO I = 1 TO 7;
  D_HUE(I) = HUE(I);
  D_LIGHT(I) = LIGHT(I);
  D_SATUR(I) = SATUR(I);
END;

/*****************************************************************/
/* Draw & fill the polygons with colors                          */
/*****************************************************************/
CALL FSINT;                       /* Initialize GDDM             */
CALL FSPCRT(99,24,80,0);          /* Create and select page 99   */
CALL GSWIN(1,80,1,24);            /* Set the window              */
CALL GSSEG(SEG_ID);               /* Create a segment            */
START_X = 16;                     /* Initialize starting x coord */
END_X = 24;                       /* Initialize ending x coord   */

DO I = 1 TO 7;
  CALL GSCOL(I);
  CALL GSAREA(1);
  CALL GSMOVE(START_X,9);
  CALL GSLINE(START_X,18);
  CALL GSLINE(END_X,18);
  CALL GSLINE(END_X,9);
  CALL GSLINE(START_X, 9);
```

```
  CALL GSENDA;

  START_X = START_X + INCR_X;
  END_X = END_X + INCR_X;
END;

CALL GSSCLS;
CALL FSFRCE;

/*****************************************************************/
/* Open the display file, then loop through showing color table  */
/* changes until CF1 is pressed.                                 */
/*****************************************************************/
OPEN FILE(CFILE) UPDATE TITLE('COLORS');
DO UNTIL(1=2);
  WRITE FILE(CFILE) OPTIONS(RECORD('COLORS')) FROM(DSPREC);
  READ FILE(CFILE) INTO(DSPREC) OPTIONS(INDICATORS(CFKEYS));

  /*****************************************************************/
  /* If CF1 is pressed, end the program.                           */
  /*****************************************************************/
  IF SUBSTR(CFKEYS,1,1)='1' THEN GO TO ENDIT;
  ELSE
    DO;

      /*************************************************************/
      /* If CF2 is pressed, redisplay the default values.          */
      /*************************************************************/
      IF SUBSTR(CFKEYS,2,1)= '1' THEN
        DO I = 1 TO 7;
          HUE(I) = S_HUE(I);
          LIGHT(I) = S_LIGHT(I);
          SATUR(I) = S_SATUR(I);
          D_HUE(I) = S_HUE(I);
          D_LIGHT(I) = S_LIGHT(I);
          D_SATUR(I) = S_SATUR(I);
        END;

      /*************************************************************/
      /* If ENTER is pressed, use the assigned values.             */
      /*************************************************************/
      ELSE
        DO I = 1 TO 7;
          HUE(I) = D_HUE(I);
          LIGHT(I) = D_LIGHT(I);
          SATUR(I) = D_SATUR(I);
        END;

      /*************************************************************/
      /* Redefine and select the new color table.                  */
      /*************************************************************/
      CALL GSCTD(65,1,7,HUE,LIGHT,SATUR);
      CALL GSCT(65);
      CALL FSFRCE;
    END;
END;
ENDIT:
CALL FSPDEL(99);
CLOSE FILE(CFILE);
END;
```

PL/I GDDM Order Form Application

This is an example of graphics program used to enhance a menu. While this example does not contain programming for managing database files, a business application modeled on this program could be used to update files on your system:




```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     A*********************************************************************
     A*
     A* TITLE: ORDERF
     A*
     A* DESCRIPTION: Display file used with the PL/I-GDDM order entry
     A*              application program 'ORDER'.
     A*
     A*********************************************************************
     A*
     A                                     DSPSIZ(24 80 *DS3)
     A                                     INDARA
     A          R REC1
     A*********************************************************************
     A*
     A* NOTE: The file or the record format(s) that are displayed
     A*       concurrently with graphics must use the ALWGPH keyword.
     A*
     A*********************************************************************
     A                                     ALWGPH
     A*********************************************************************
     A*
     A* NOTE: The field names shown below are automatically declared in
     A*       the PL/I program by the %INCLUDE function.
     A*
     A*********************************************************************
```

```
A            ORDERNUM      9A  B  8 14
A            REQUESTER    14A  B  8 27
A            BUYER         5A  B  8 45
A            DEPTCHRG      5A  B  8 54
A            TC            2A  B  8 63
A            CHECK         1A  B  8 69
A            DATEREQ       7A  B 14 13
A            USAGECODE    19A  B 14 22
A            REFNUMBER    11A  B 14 43
A            JK            1A  B 14 56
A            RB            1A  B 14 59
A            BS            1A  B 14 62
A            DEPTDLVR      5A  B 14 65
```

What follows is the PL/I program you can use to produce the picture. This is an example program, and so the function key mentioned does not do anything. It is there to give you an idea of the type of system that you can set up.

```
ORDER: PROCEDURE;
/*********************************************************************/
/*  Declare the external display file used for the input fields      */
/*  and include the field names in a structure.                      */
/*********************************************************************/
DCL
  ALPHA_FILE  FILE  RECORD  ENV(INTERACTIVE)  UPDATE;
DCL
  1 ALPHA_BUFFER  AUTOMATIC,
    %INCLUDE  ORDERF(REC1,RECORD);
/*********************************************************************/
/*  Declare the variables used with GSQCTB.                          */
/*********************************************************************/
DCL
  XARRAY  (5)  FLOAT  DEC(6),
  YARRAY  (5)  FLOAT  DEC(6);
/*********************************************************************/
/*  Declare the variables used with GSQCB.                           */
/*********************************************************************/
DCL
  DEFAULT_CBOXX  FLOAT  DEC(6),
  DEFAULT_CBOXY  FLOAT  DEC(6),
  NEW_CBOXX      FLOAT  DEC(6),
  NEW_CBOXY      FLOAT  DEC(6);
/*********************************************************************/
/*  Declare temporary working variables.                             */
/*********************************************************************/
DCL
  TEMP_FLOAT  FLOAT DEC(6),
  TEMP_FLOAT2 FLOAT DEC(6),
  NEW_LINE    CHAR(1),        /* Contains hex 15 new line
                                 control character.                */
  TEMP_TEXT   CHAR(60);       /* Character string variable.        */
/*********************************************************************/
/*  Initialize GDDM                                                  */
/*********************************************************************/
CALL FSINIT;
                              /* Default field is the entire
                                 screen area.                      */
                              /* Default picture space is the
                                 entire screen area.               */
                              /* Default viewport is the entire
                                 screen area.                      */
```

```
/*********************************************************************/
/* Use a window that corresponds to the alphameric display           */
/* dimensions of 80 columns by 24 rows, which corresponds to         */
/* external display file dimensions.                                 */
/*********************************************************************/
CALL GSWIN (1,81,1,25);
/*********************************************************************/
/*  Create a segment.                                                */
/*********************************************************************/
CALL GSSEG (1);
/*********************************************************************/
/*  Enclose the screen in a frame.                                   */
/*********************************************************************/
CALL GSCOL (1);              /* 1 = blue.                            */
CALL GSLW (2);               /* 2 = double-width line.               */
CALL GSMOVE (1,1);
CALL GSLINE (81,1);
CALL GSLINE (81,25);
CALL GSLINE (1,25);
CALL GSLINE (1,1);
/*********************************************************************/
/*  Draw the title, centered and underlined.                         */
/*********************************************************************/
CALL GSCOL (6);              /* 6 = yellow.                          */
CALL GSCHAR (28.5,23.50,23,'SHOP ORDER ENTRY SCREEN');
CALL GSQTB (23,'SHOP ORDER ENTRY SCREEN',5,XARRAY,YARRAY);
TEMP_FLOAT = 28.5 + XARRAY (4);
CALL GSLW (1);               /* 1 = standard-width line.             */
CALL GSMOVE (28.5,23.25);
CALL GSLINE (TEMP_FLOAT,23.25);    /*  Underline.                     */
/*********************************************************************/
/*  Draw the primary box around the input fields.                    */
/*********************************************************************/
CALL GSCOL (1);              /* 1 = blue.                            */
CALL GSMOVE (8.50,22.50);    /* Upper left corner.                   */
CALL GSLINE (72.25,22.50);   /* Draw to upper right.                 */
CALL GSLINE (72.25,10.00);   /* Draw to lower right.                 */
CALL GSLINE (8.50,10.00);    /* Draw to lower left.                  */
CALL GSLINE (8.50,22.50);    /* Draw to upper left.                  */
/*********************************************************************/
/*  Draw the first line of horizontal character text.                */
/*********************************************************************/
CALL GSCOL (2);              /* RED.                                 */
CALL GSCHAR (14.00,20.00,10,'SHOP ORDER');
CALL GSCHAR (27.00,20.00,9,'REQUESTED');
CALL GSCHAR (45.00,20.00,5,'BUYER');
CALL GSCHAR (54.00,20.00,4,'DEPT');
/*********************************************************************/
/*  Draw the second line of horizontal character text.               */
/*********************************************************************/
CALL GSCHAR (14.00,19.00,6,'NUMBER');
CALL GSCHAR (30.00,19.00,2,'BY');
CALL GSCHAR (45.00,19.00,4,'CODE');
CALL GSCHAR (53.00,19.00,7,'CHARGED');
CALL GSCHAR (62.50,19.00,3,'T/C');
/*********************************************************************/
/*  Draw the third line of horizontal character text.                */
/*********************************************************************/
CALL GSCHAR (15.00,14.00,4,'DATE');
CALL GSCHAR (24.00,13.50,11,'USAGE CODES');
```

```
CALL GSCHAR (43.00,14.00,9,'REFERENCE');
CALL GSCHAR (56.00,14.10,6,'PREFIX');
CALL GSCHAR (64.50,14.00,7,'DELIVER');
/*********************************************************************/
/*  Draw the fourth line of horizontal character text.               */
/*********************************************************************/
CALL GSCHAR (13.00,13.00,8,'REQUIRED');
CALL GSCHAR (43.00,13.00,6,'NUMBER');
CALL GSCHAR (55.50,13.00,2,'JK');
CALL GSCHAR (58.50,13.00,2,'RB');
CALL GSCHAR (61.50,13.00,2,'BS');
CALL GSCHAR (64.50,13.00,7,'TO DEPT');
/*********************************************************************/
/*  Draw the 'notes' section at the bottom of the screen.            */
/*********************************************************************/
CALL GSCOL (7);                 /* 7 = white.                        */
CALL GSCHAR (14.0,7.0,5,'NOTES:');
CALL GSCHAR (16.0,5.5,30,'1. Use same number entered on ');
CALL GSQTB  (30,'Use same number entered on ',5,XARRAY,YARRAY);
TEMP_FLOAT = 16.0 + XARRAY (4); /* Compute starting position to
                                   underline the word 'written'.    */
CALL GSCHAR (TEMP_FLOAT,5.5,7,'written');
CALL GSQTB  (7,'written',5,XARRAY,YARRAY);
TEMP_FLOAT2 = TEMP_FLOAT + XARRAY (4); /* Compute ending position of
                                   underline (also the start of the
                                   remainder of the string).        */
CALL GSLW (1);                 /* 1 = standard=width line.          */
CALL GSMOVE (TEMP_FLOAT,5.4);  /* Move to start of underline.       */
CALL GSLINE (TEMP_FLOAT2,5.4); /* Draw the underline.               */
CALL GSCHAR (TEMP_FLOAT2,5.5,9,' request.'); /* Finish sentence.    */
CALL GSCHAR (16.0,4.0,37,'2.  F13: Table of current references.');
/*********************************************************************/
/*  Draw the first horizontal line.                                  */
/*********************************************************************/
CALL GSCOL (1);                /* BLUE.                             */
CALL GSMOVE (12.75,22.25);
CALL GSLINE (72.25,22.25);
/*********************************************************************/
/*  Draw the second horizontal line.                                 */
/*********************************************************************/
CALL GSMOVE (12.75,21.25);
CALL GSLINE (67.50,21.25);
/*********************************************************************/
/*  Draw the third horizontal line.                                  */
/*********************************************************************/
CALL GSMOVE (12.75,18.75);
CALL GSLINE (72.25,18.75);
/*********************************************************************/
/*  Draw the fourth and fifth horizontal lines.                      */
/*********************************************************************/
CALL GSMOVE (12.75,16.50);
CALL GSLINE (72.25,16.50);
CALL GSMOVE (12.75,16.25);
CALL GSLINE (72.25,16.25);
/*********************************************************************/
/*  Draw the sixth and seventh horizontal lines.                     */
/*********************************************************************/
CALL GSMOVE (12.75,15.25);
CALL GSLINE (72.25,15.25);
CALL GSMOVE (12.75,12.75);
```

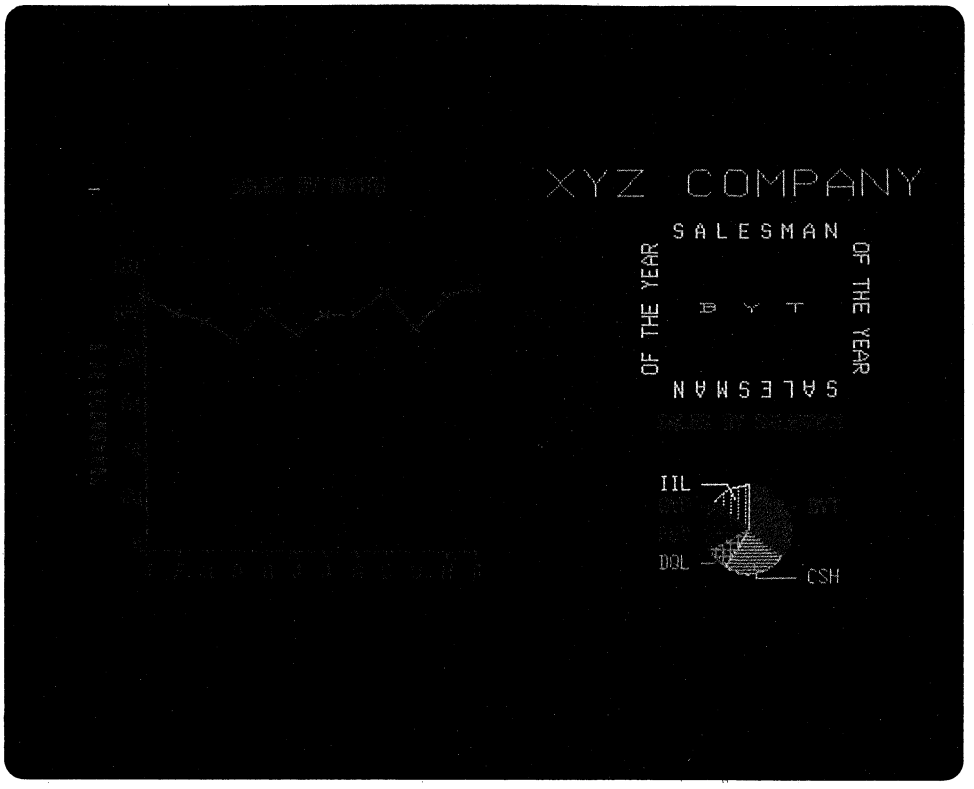
```
CALL GSLINE (72.25,12.75);
/*********************************************************************/
/*  Draw the horizontal line under 'PREFIX'.                         */
/*********************************************************************/
CALL GSMOVE (55.25,14.10);
CALL GSLINE (64.25,14.10);
/*********************************************************************/
/*  Draw the left-most interior vertical line.                       */
/*********************************************************************/
CALL GSMOVE (12.75,22.50);
CALL GSLINE (12.75,10.00);
/*********************************************************************/
/*  Draw the remaining vertical lines in the top half of the form.   */
/*********************************************************************/
CALL GSMOVE (25.50,22.25);
CALL GSLINE (25.50,16.50);
CALL GSMOVE (43.50,22.25);
CALL GSLINE (43.50,16.50);
CALL GSMOVE (52.00,22.25);
CALL GSLINE (52.00,16.50);
CALL GSMOVE (62.00,22.25);
CALL GSLINE (62.00,16.50);
CALL GSMOVE (67.50,22.25);
CALL GSLINE (67.50,16.50);
/*********************************************************************/
/*  Draw the vertical lines in the lower half of the form.           */
/*********************************************************************/
CALL GSMOVE (21.50,16.25);
CALL GSLINE (21.50,10.00);
CALL GSMOVE (42.50,16.25);
CALL GSLINE (42.50,10.00);
CALL GSMOVE (55.25,16.25);
CALL GSLINE (55.25,10.00);
CALL GSMOVE (64.25,16.25);
CALL GSLINE (64.25,10.00);
/*********************************************************************/
/*  Draw the two dotted vertical lines that separate 'JK', 'RB' and  */
/*  'BS'.                                                            */
/*********************************************************************/
CALL GSLT (1);               /* 1 = dotted line.                     */
CALL GSMOVE (58.00,12.75);
CALL GSLINE (58.00,10.00);
CALL GSMOVE (61.00,12.75);
CALL GSLINE (61.00,10.00);
/*********************************************************************/
/*  Query the character box size to enable proportional character    */
/*  size reduction.                                                  */
/*********************************************************************/
CALL GSQCB (DEFAULT_CBOXX,DEFAULT_CBOXY);
/*********************************************************************/
/*  Select character mode 3.                                         */
/*********************************************************************/
CALL GSCM (3);
/*********************************************************************/
/*  Draw the small word 'CHECK' at a 90-degree character angle.      */
/*********************************************************************/
CALL GSCOL (2);                  /* 2 = red.                         */
NEW_CBOXX = DEFAULT_CBOXX * .65; /* New x size is 65% of default.    */
NEW_CBOXY = DEFAULT_CBOXY;       /* y size remains same as default. */
CALL GSCB (NEW_CBOXX,NEW_CBOXY);
```

```
CALL GSCA (0.0,1.0);              /* Set 90-degree character angle.  */
CALL GSCHAR (70.00,18.85,5,'CHECK');
/*********************************************************************/
/*  Draw the vertical characters at the left side of the figure.     */
/*********************************************************************/
CALL GSCOL (7);                   /* 7 = white.                      */
UNSPEC(NEW_LINE) = '00010101'B;   /* Use 'new line' control          */
TEMP_TEXT = 'COMPLETE ALL PARTS' ||
            NEW_LINE                ||
            'BEFORE SUBMITTING';
CALL GSCHAR (10.50,10.50,36,TEMP_TEXT);
/*********************************************************************/
/*  Draw the small numbers.                                          */
/*********************************************************************/
CALL GSCA (1.0,0.0);              /* Restore normal character angle. */
NEW_CBOXX = DEFAULT_CBOXX * .80;  /* New x size is 80% of default.   */
NEW_CBOXY = DEFAULT_CBOXY * .90;  /* New y size is 90% of default.   */
CALL GSCB (NEW_CBOXX,NEW_CBOXY);
CALL GSCOL (1);                   /* 1 = blue.                       */
CALL GSCHAR (13.25,21.25,1,'1');
CALL GSCHAR (24.50,21.25,1,'9');
CALL GSCHAR (26.00,21.25,2,'10');
CALL GSCHAR (41.25,21.25,2,'23');
CALL GSCHAR (44.25,21.25,2,'24');
CALL GSCHAR (50.25,21.25,2,'28');
CALL GSCHAR (52.50,21.25,2,'29');
CALL GSCHAR (60.25,21.25,2,'33');
CALL GSCHAR (62.50,21.25,2,'34');
CALL GSCHAR (65.25,21.25,2,'35');
CALL GSCHAR (13.25,15.25,2,'36');
CALL GSCHAR (19.75,15.25,2,'42');
CALL GSCHAR (22.00,15.25,2,'43');
CALL GSCHAR (40.75,15.25,2,'61');
CALL GSCHAR (43.00,15.25,2,'62');
CALL GSCHAR (53.50,15.25,2,'72');
CALL GSCHAR (55.75,15.25,2,'73');
CALL GSCHAR (62.50,15.25,2,'75');
CALL GSCHAR (64.75,15.25,2,'76');
CALL GSCHAR (70.50,15.25,2,'80');
/*********************************************************************/
/*  Draw the small 'NOTE' numbers.                                   */
/*********************************************************************/
CALL GSCOL (7);                 /* 7 = white.                        */
CALL GSCHAR (22.50,19.00,1,'1');
CALL GSCHAR (51.50,13.00,1,'2');
CALL FSFRCE;
/*********************************************************************/
/*  The order form has now been drawn.                               */
/*  Open the ALPHA_FILE and perform a read/write operation to        */
/*  accept the input.                                                */
/*********************************************************************/
OPEN FILE(ALPHA_FILE)  UPDATE  TITLE('ORDERF');
/*********************************************************************/
/* Set the initial values to the record format field names.          */
/* NOTE: The field variables were automatically declared by the      */
/*       '%INCLUDE' function near the start of the program.          */
/*********************************************************************/
ORDERNUM = 'XXXXXXXXX';
REQUESTER = 'XXXXXXXXXXXXXX';
BUYER = 'XXXXX';
```

```
DEPTCHRG = 'XXXXX';
TC = 'XX';
CHECK = 'X';
DATEREQ = 'XXXXXXX';
USAGECODE = 'XXXXXXXXXXXXXXXXXX';
REFNUMBER = 'XXXXXXXXXX';
JK = 'XX';
RB = 'XX';
BS = 'XX';
DEPTDLVR = 'XXXXX';
/*********************************************************************/
/* Perform the read/write operation(s).                              */
/* NOTE: This can be done many times without redrawing the order form*/
/*********************************************************************/
WRITE FILE(ALPHA_FILE) FROM(ALPHA_BUFFER) OPTIONS(RECORD('REC1'));
READ FILE(ALPHA_FILE)  INTO(ALPHA_BUFFER);
/*********************************************************************/
/*  Close the display file and terminate GDDM.                       */
/*********************************************************************/
CLOSE FILE(ALPHA_FILE);
CALL FSTERM;
/*********************************************************************/
/*  Include the GDDM entry declarations.                             */
/*********************************************************************/
%INCLUDE SYSLIB(ADMUPLNB);
END ORDER;
```

RPG/400 Program with Presentation Graphics and GDDM

This is an example of graphics program written in the RPG/400 programming language that uses both Presentation Graphics and GDDM routines:




```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     H* This RPG program shows GDDM and Presentation Graphics routines
     H* being used in the same RPG program.  The first part of the
     H* program draws a pie chart and a line chart; the second part
     H* uses GDDM to draw a picture.
     H        1
     FPARSUMRYIF  E                    DISK
     E* The following declarations for floating point arrays;  those
     E* shown with a blank in column 35 are execution-time arrays.
     E                    AX     12  12  2 0            X VALUE FOR LINE
     E                    AY         12 10 2            Y VALUE FOR LINE
     E                    TTL        20 10 2            TOTAL BY SLSMN
     E* The following declarations are for 4-byte binary integer arrays,
     E* which must be defined as a DS on an input spec.
     E                    HATT    1   4  9 0            HEADING ATTR
     E                    AATT    1   4  9 0            AXIS ATTR
     E                    KATT    1   4  9 0            KEY ATTR
     E                    LATT    1   4  9 0            LABEL ATTR
     E* The following character array is used for up to 20 pie chart
     E* labels.  The array elements are obtained by program calculations.
     E                    PLAB       20  3              LABELS FOR PIE
     E* These single-element character arrays are used for heading text
     E* and axis titles.  The character strings are too long to be moved
     E* into a field as character literals in factor 2.
     E                    LINHED  1   1 14              HEAD FOR LINE
     E                    YTTL    1   1 26              Y AXIS TITLE
     E                    PIEHED  1   1 17              HEAD FOR PIE
```

```
E                    TSALE   1   1 30              TEXT FOR GDDM
E                    TYEAR   1   1 30              TEXT FOR GDDM
I* The following integer arrays are defined as binary arrays.
IARRAYS      DS
I                                        B   1  160HATT
I                                        B  17  320AATT
I                                        B  33  480KATT
I                                        B  49  640LATT
I* The following declaration statements are for other 4-byte
I* binary integer variables.
IPARMS       DS
I                                        B   1   40DTAGRP
I                                        B   5   80COUNT
I                                        B   9  120LENGTH
I                                        B  13  160NUM
I                                        B  17  200INDEX
I                                        B  21  240ATTYPE
I                                        B  25  280ATMOD
I                                        B  29  320CNT
C* The following section does general housekeeping;
C* 'C' is a counter used in the program.
C                    Z-ADD0          C       20
C           1        DO   12         C
C                    Z-ADD0          AY,C
C                    END  1
C                    Z-ADD1          C
C                    MOVEA*BLANKS    PLAB,C
C                    Z-ADD0          C
C           1        DO   20         C
C                    Z-ADD0          TTL,C
C                    END  1
C                    Z-ADD0          C
C* NAMSAV saves names for use in pie labels.
C                    MOVE *BLANKS    NAMSAV  3
C* MONTH is used when accumulating sales figures for each month.
C                    Z-ADD0          MONTH   60
C* I IS ANOTHER COUNTER
C                    Z-ADD0          I       20
C* INDEX contains the number of labels used for the pie chart.
C                    Z-ADD0          INDEX
C* NUM is used in many routines where an integer is needed.
C                    Z-ADD0          NUM
C* The following variables are used to set the viewport and window
C* for use by the GDDM routines.
C                    Z-ADD0          LEFT    63
C                    Z-ADD0          RIGHT   63
C                    Z-ADD0          LOWER   63
C                    Z-ADD0          UPPER   63
C                    Z-ADD0          WIDTH   63
C                    Z-ADD0          DEPTH   63
C* The following section reads file PARSUMRY and calculates the
C* data needed for the line chart and the pie chart.
C* In factor one in the TAG statement, RED = read and FIL = file.
C                    Z-ADD0          C
C           REDFIL   TAG
C                    READ PARSUMRY                  99
C   99               GOTO EOF
C           SLSMAN   CABEQNAMSAV     SUMTOT
C                    MOVE SLSMAN     NAMSAV
C                    ADD  1          C
```

```
C                   MOVE SLSMAN    PLAB,C
C           SUMTOT  TAG
C                   ADD  OTOTAL    TTL,C
C* In file PARSUMRY, 'INVDT' is the invoice date used to
C* accumulate sales by month;  its format is MMDDYY.
C* MONTH compares and adds the sales values into
C* the correct array element for the y-values for the line chart.
C           1       DO   12        I
C           I       MULT 10000     MONTH
C                   ADD  10000     MONTH
C           INVDT   CABLTMONTH     TOTLIN
C                   END  1
C                   GOTO REDFIL
C           TOTLIN  TAG
C                   ADD  OTOTAL    AY,I
C                   GOTO REDFIL
C           EOF     TAG
C                   Z-ADDC         INDEX
C                   Z-ADD0         C
C           1       DO   12        C
C                   MULT .001      AY,C
C                   END  1
C* ******************************************************
C*              PRESENTATION GRAPHICS SECTION
C* THIS SECTION DRAWS THE LINE CHART
C                   CALL 'GDDM'                       INITIALIZE
C                   PARM 'FSINIT  'FSINIT  8            GRAPHICS
C                   CALL 'GDDM'
C                   PARM 'GSQPS   'GSQPS   8          QUERY PIC
C                   PARM           WIDTH                SPACE
C                   PARM           DEPTH
C                   Z-ADD0         LEFT
C           .5      MULT WIDTH     RIGHT
C                   Z-ADD0         LOWER
C                   Z-ADDDEPTH     UPPER
C                   CALL 'GDDM'                       CHART DRAWN
C                   PARM 'CHAREA  'CHAREA  8            AT LEFT
C                   PARM           LEFT                 SIDE OF
C                   PARM           RIGHT                SCREEN
C                   PARM           LOWER
C                   PARM           UPPER
C                   CALL 'GDDM'                       HEADING FOR
C                   PARM 'CHHEAD  'CHHEAD  8            LINE CHART
C                   PARM 14        LENGTH
C                   PARM           LINHED
C                   CALL 'GDDM'                       SET HEADING
C                   PARM 'CHHATT  'CHHATT  8            ATTRIBUTES
C                   PARM 4         COUNT
C                   PARM           HATT
C                   CALL 'GDDM'                       MONTHS STRT
C                   PARM 'CHXMTH  'CHXMTH  8            WITH JAN
C                   PARM 1         NUM
C                   CALL 'GDDM'                       NOFORCEZERO
C                   PARM 'CHXSET  'CHXSET  8
C                   PARM 'NOFO'    NOFRCZ  4
C                   CALL 'GDDM'                       USE FIRST
C                   PARM 'CHSET   'CHSET   8            LETTER
C                   PARM 'LETTER'  LETTER  6
C                   CALL 'GDDM'                       SET THE AXIS
C                   PARM 'CHTATT  'CHTATT  8            TITLE
```

```
C                   PARM 4         COUNT             ATTRIBUTES
C                   PARM           AATT
C                   CALL 'GDDM'                      Y AXIS TITLE
C                   PARM 'CHYTTL   'CHYTTL  8
C                   PARM 26        LENGTH
C                   PARM           YTTL
C                   CALL 'GDDM'                      SET AXIS
C                   PARM 'CHLATT   'CHLATT  8          LABEL
C                   PARM 4         COUNT               ATTRIBUTES
C                   PARM           LATT
C                   CALL 'GDDM'                      DRAW THE LINE
C                   PARM 'CHPLOT   'CHPLOT  8          CHART
C                   PARM 1         DTAGRP
C                   PARM 12        COUNT
C                   PARM           AX
C                   PARM           AY
C* THIS SECTION DRAWS THE PIE CHART
C                   CALL 'GDDM'                      RESTART GRAPHICS
C                   PARM 'CHSTRT   'CHSTRT  8
C           .5      MULT WIDTH     LEFT
C                   Z-ADDWIDTH     RIGHT
C                   Z-ADD0         LOWER
C           .5      MULT DEPTH     UPPER
C                   CALL 'GDDM'
C                   PARM           CHAREA            SET CHART AREA
C                   PARM           LEFT
C                   PARM           RIGHT
C                   PARM           LOWER
C                   PARM           UPPER
C                   CALL 'GDDM'                      HEADING FOR
C                   PARM           CHHEAD              PIE CHART
C                   PARM 17        LENGTH
C                   PARM           PIEHED
C                   CALL 'GDDM'                      HEADING ATTR
C                   PARM           CHHATT
C                   PARM 4         COUNT
C                   PARM           HATT
C                   CALL 'GDDM'                      SUPPRESS LBLS
C                   PARM           CHXSET
C                   PARM 'NOLAB'   NOLAB   5
C                   CALL 'GDDM'                      KEY ATTR
C                   PARM 'CHKATT   'CHKATT  8
C                   PARM 4         COUNT
C                   PARM           KATT
C                   CALL 'GDDM'                      USE SPIDER
C                   PARM 'CHSET    'CHSET   8          LABELS
C                   PARM 'SPIDER'  SPIDER  6
C                   CALL 'GDDM'                      LABELS FOR
C                   PARM 'CHKEY    'CHKEY   8          PIE CHART
C                   PARM           INDEX
C                   PARM 3         LENGTH
C                   PARM           PLAB
C                   CALL 'GDDM'                      DRAW COMPLETE
C                   PARM           CHSET               PIE
C                   PARM 'ABPIE'   ABPIE   5
C                   CALL 'GDDM'
C                   PARM 'CHPIER   'CHPIER  8        REDUCE PIE
C                   PARM 50        NUM                 50%
C                   CALL 'GDDM'
C                   PARM 'CHPIE    'CHPIE   8        DRAW PIE
```

```
C                   PARM 1          DTAGRP
C                   PARM            INDEX
C                   PARM            TTL
C                   CALL 'GDDM'                         TERMINATE GRAPHICS
C                   PARM 'CHTERM  'CHTERM  8
C* ****************************************************
C*                     GDDM SECTION
C* THE VIEWPORT IS SET TO THE RIGHT HALF OF THE SCREEN
C           .5      MULT WIDTH      LEFT
C                   Z-ADDWIDTH      RIGHT
C           .5      MULT DEPTH      LOWER
C                   Z-ADDDEPTH      UPPER
C                   CALL 'GDDM'
C                   PARM 'GSVIEW  'GSVIEW  8
C                   PARM            LEFT
C                   PARM            RIGHT
C                   PARM            LOWER
C                   PARM            UPPER
C* THE WINDOW IS SET TO X=0-100 AND Y=0-100
C                   Z-ADD0          LEFT
C                   Z-ADD100        RIGHT
C                   Z-ADD0          LOWER
C                   Z-ADD100        UPPER
C                   CALL 'GDDM'
C                   PARM 'GSWIN   'GSWIN   8
C                   PARM            LEFT
C                   PARM            RIGHT
C                   PARM            LOWER
C                   PARM            UPPER
C                   CALL 'GDDM'
C                   PARM 'GSCM    'GSCM    8
C                   PARM 3          NUM
C                   CALL 'GDDM'
C                   PARM 'GSCOL   'GSCOL   8
C                   PARM 7          NUM
C                   CALL 'GDDM'
C                   PARM 'GSCHAR  'GSCHAR  8
C                   PARM 30         XVAL    41
C                   PARM 70         YVAL    41
C                   PARM 17         LENGTH
C                   PARM            TSALE
C                   CALL 'GDDM'
C                   PARM 'GSCB    'GSCB    8
C                   PARM 5          XVAL
C                   PARM 5          YVAL
C                   CALL 'GDDM'
C                   PARM 'GSCA    'GSCA    8
C                   PARM 0          XVAL
C                   PARM -1         YVAL
C                   CALL 'GDDM'
C                   PARM 'GSCHAP  'GSCHAP  8
C                   PARM 11         LENGTH
C                   PARM            TYEAR
C                   CALL 'GDDM'
C                   PARM 'GSCB    'GSCB
C                   PARM 2.5        XVAL
C                   PARM 10         YVAL
C                   CALL 'GDDM'
C                   PARM            GSCA
C                   PARM -1         XVAL
```

```
     C                   PARM 0         YVAL
     C                   CALL 'GDDM'
     C                   PARM           GSCHAP
     C                   PARM 17        LENGTH
     C                   PARM           TSALE
     C                   CALL 'GDDM'
     C                   PARM 'GSCB    'GSCB
     C                   PARM 5         XVAL
     C                   PARM 5         YVAL
     C                   CALL 'GDDM'
     C                   PARM           GSCA
     C                   PARM 0         XVAL
     C                   PARM 1         YVAL
     C                   CALL 'GDDM'
     C                   PARM           GSCHAP
     C                   PARM 11        LENGTH
     C                   PARM           TYEAR
     C                   CALL 'GDDM'
     C                   PARM 'GSCA    'GSCA    8
     C                   PARM 1         XVAL
     C                   PARM 0         YVAL
     C                   CALL 'GDDM'
     C                   PARM           GSCOL
     C                   PARM 2         NUM
     C                   CALL 'GDDM'
     C                   PARM           GSCHAR
     C                   PARM 33        XVAL
     C                   PARM 40        YVAL
     C                   PARM 7         LENGTH
     C                   PARM ' B Y T ' MANYR   7
     C                   CALL 'GDDM'
     C                   PARM 'GSCB    'GSCB    8
     C                   PARM 8         XVAL
     C                   PARM 16        YVAL
     C                   CALL 'GDDM'
     C                   PARM           GSCHAR
     C                   PARM 3         XVAL
     C                   PARM 85        YVAL
     C                   PARM 4         LENGTH
     C                   PARM 'XYZ '    HEAD1   4
     C                   CALL 'GDDM'
     C                   PARM           GSCHAP
     C                   PARM 8         LENGTH
     C                   PARM 'COMPANY 'HEAD2   8
     C                   CALL 'GDDM'
     C                   PARM 'ASREAD  'ASREAD  8
     C                   PARM           ATTYPE
     C                   PARM           ATMOD
     C                   PARM           CNT
     C                   CALL 'GDDM'
     C                   PARM 'FSTERM  'FSTERM  8
     C                   SETON                    LR
     C                   RETRN
**
0102030405060708091011l2
**
000000005
000000002
000000000
000000200
```

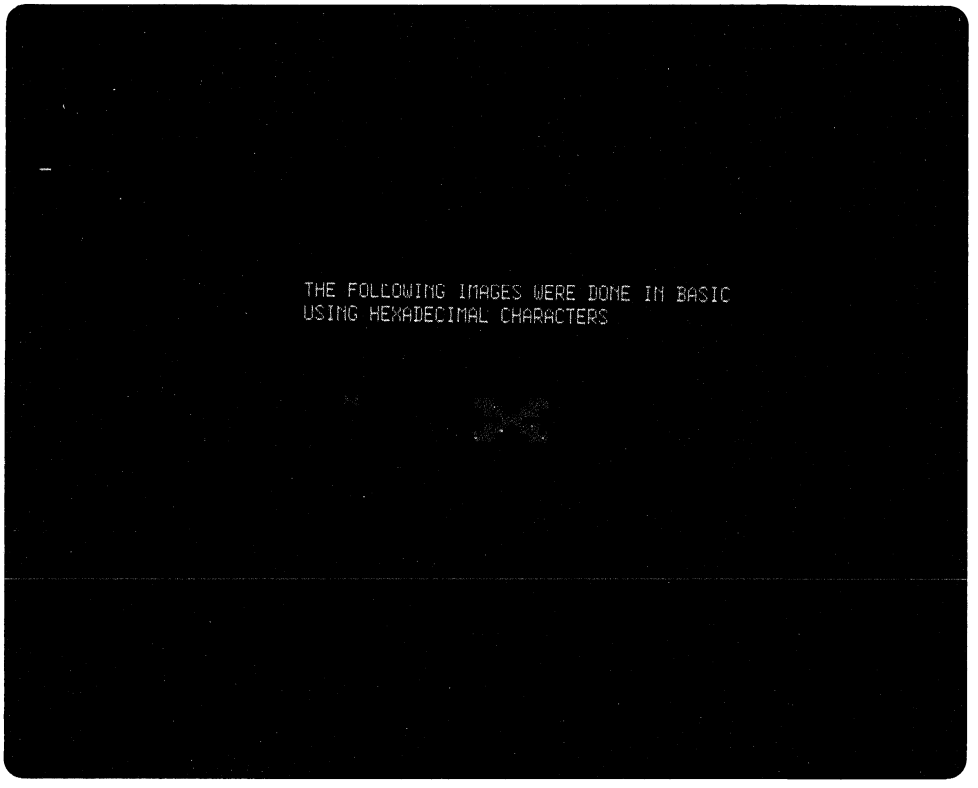
```
**
000000005
000000003
000000000
000000100
**
000000000
000000002
000000000
000000200
**
000000004
000000002
000000000
000000200
**
SALES BY MONTH
**
T H O U S A N D S  O F  $
**
SALES BY SALESMAN
**
 S A L E S M A N
**
OF THE YEAR
```

Graphics Image Programs in Each Language

The programs in this section show how a graphics image and the same graphics image *scaled* are drawn by each of the high-level languages.

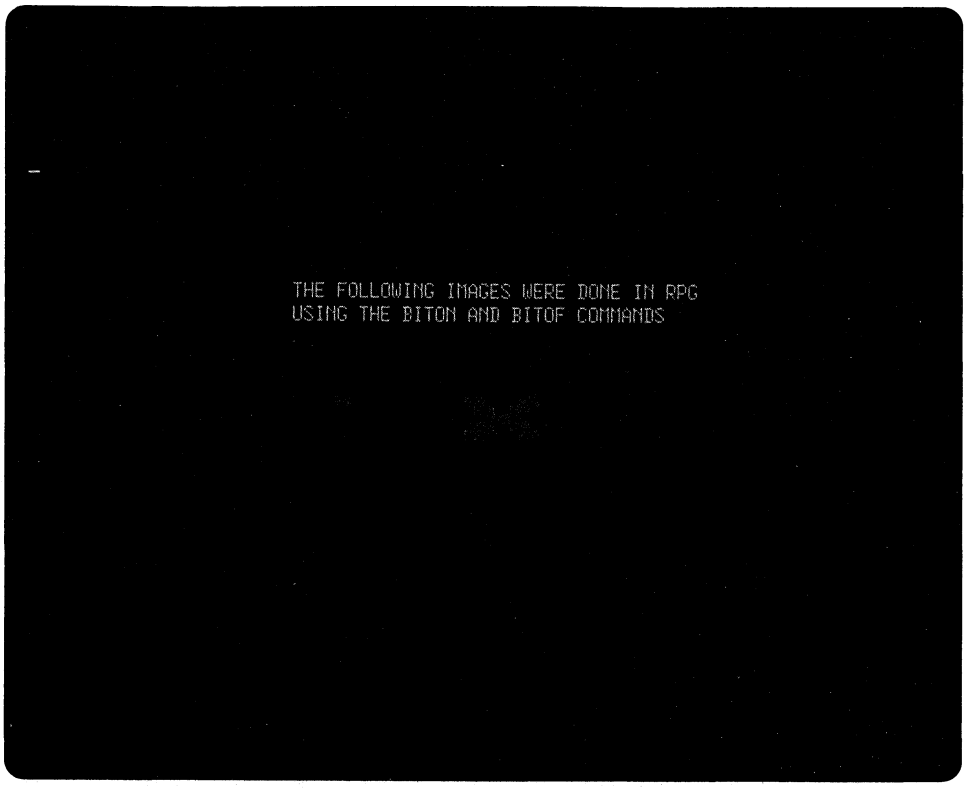
Graphics images are constructed by programs that convert data given in *bit patterns* to corresponding pictures, where each *on* bit sets a pixel on, and each *off* bit leaves the pixel as it was.

Graphics Image Drawn in BASIC




```
00010 CALL GDDM ('FSINIT')              ! Initialize graphics
00020 CALL GDDM ('GSCOL',2)             ! Set color
00030 DIM CHAR$*39 : CHAR$ = 'THE FOLLOWING IMAGES WERE DONE IN BASIC'
00040 CALL GDDM ('GSCHAR',30.0,70.0,39,CHAR$)
00050 CALL GDDM ('GSCHAR',30.0,65.0,29,'USING HEXADECIMAL CHARACTERS')
00060 CALL GDDM ('GSCOL',1)             ! Set color
00070 CALL GDDM ('GSMOVE',35.0,50.0)    ! Move to image position
00080 DIM IMG$*5                        ! Image variable 1/2 hex string
00090 DIM X$*10                         ! Hex string variable
00100 X$='C3661866C3'
00110 IMG$=HEX$(X$)                     ! Convert back from hexadecimal
00120 CALL GDDM ('GSIMG',0,8,5,5,IMG$)
00130 ! Draw image with 8 pixels across, 5 display
00140 ! points down, 5 bytes of pixels ((8*5)/8 = total)
00150 CALL GDDM ('GSMOVE',50.0,50.0)    ! Move image position
00160 CALL GDDM ('GSIMGS',0,8,5,5,IMG$,10.0,10.0)
00170 ! Draw image scaled to 10 x-units, 10 y-units
00180 INTEGER ATTYPE,ATTVAL,COUNT
00190 CALL GDDM ('ASREAD', ATTYPE,ATTVAL,COUNT)
00200 CALL GDDM ('FSTERM')
00210 END
```

Graphics Image Drawn in the RPG/400 Programming Language

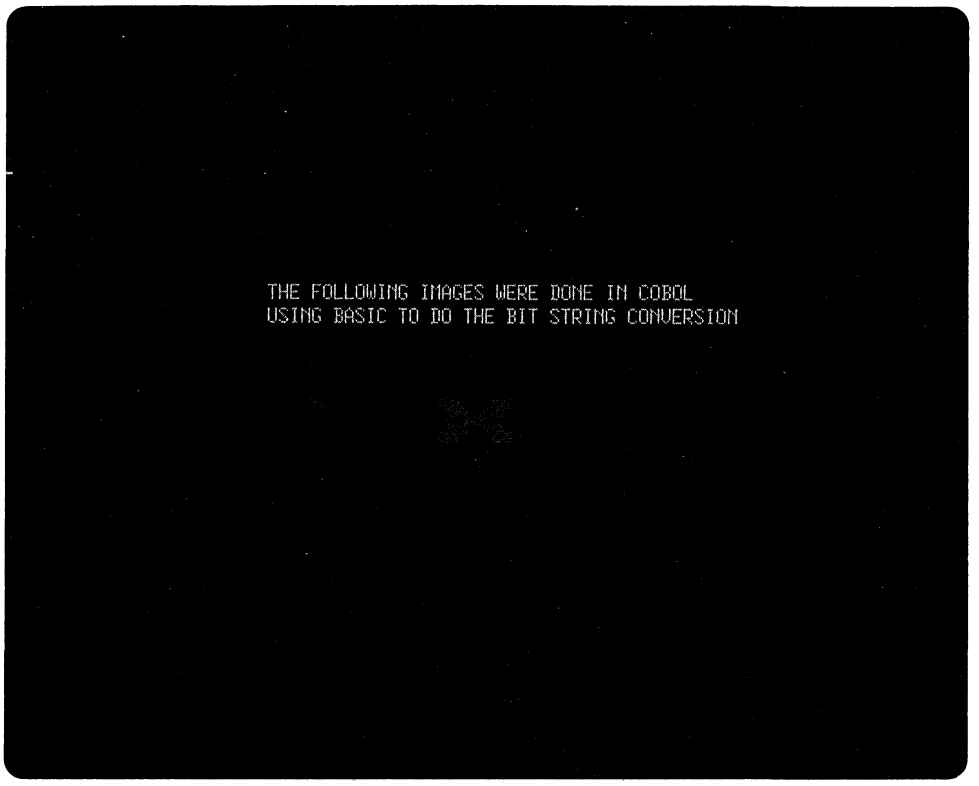



```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     E                    HEADG1  1   1 45
     E                    HEADG2  1   1 45
     E                    IMAG        8  1
     I*          Declare the image character string
     IIDTA        DS
     I                                        1   8 IMAG
     I*          Declare the image variables as binary 4
     IPARAM       DS
     I                                    B   1   40ATTYPE
     I                                    B   5   80ATTVAL
     I                                    B   9  120COUNT
     I                                    B  13  160WIDTH
     I                                    B  17  200DEPTH
     I                                    B  21  240BYTES
     I                                    B  25  280ITYPE
     I                                    B  29  320COLOR
     C*         Set off the 2nd bit of each element of IDTA,
     C*         necessary because the image data is initially
     C*         set to blanks (X'40'):
     C                    BITOF'1'       IMAG,1
     C                    BITOF'1'       IMAG,2
     C                    BITOF'1'       IMAG,3
     C                    BITOF'1'       IMAG,4
     C                    BITOF'1'       IMAG,5
     C*         After the image data is set to zeros,
     C*         use BITON to set the image point bits on:
     C                    BITON'0167'    IMAG,1
     C                    BITON'1256'    IMAG,2
     C                    BITON'34'      IMAG,3
```

```
C                   BITON'1256'    IMAG,4
C                   BITON'0167'    IMAG,5
C*          The following routines draw the image:
C                   CALL 'GDDM'                          Initialize
C                   PARM 'FSINIT'  FSINIT  8             Graphics
C                   Z-ADD2         COLOR
C                   CALL 'GDDM'                          Set color
C                   PARM 'GSCOL'   GSCOL   8
C                   PARM           COLOR
C                   Z-ADD30        XMOV    31
C                   Z-ADD70        YMOV    31
C                   Z-ADD45        COUNT
C                   CALL 'GDDM'                          Heading
C                   PARM 'GSCHAR'  GSCHAR  8
C                   PARM           XMOV
C                   PARM           YMOV
C                   PARM           COUNT
C                   PARM           HEADG1
C                   Z-ADD30        XMOV    31
C                   Z-ADD65        YMOV    31
C                   CALL 'GDDM'                          Heading
C                   PARM 'GSCHAR'  GSCHAR  8
C                   PARM           XMOV
C                   PARM           YMOV
C                   PARM           COUNT
C                   PARM           HEADG2
C                   Z-ADD1         COLOR
C                   CALL 'GDDM'                          Set color
C                   PARM 'GSCOL'   GSCOL   8
C                   PARM           COLOR
C                   Z-ADD35        XMOV    31
C                   Z-ADD50        YMOV    31
C                   CALL 'GDDM'                          Move to
C                   PARM 'GSMOVE'  GSMOVE  8             position
C                   PARM           XMOV
C                   PARM           YMOV
C                   Z-ADD0         ITYPE
C                   Z-ADD8         WIDTH
C                   Z-ADD5         DEPTH
C                   Z-ADD5         BYTES
C                   Z-ADD10        SCALX   31
C                   Z-ADD10        SCALY   31
C                   CALL 'GDDM'                           Draw image
C                   PARM 'GSIMG '  GSIMG   8
C                   PARM           ITYPE
C                   PARM           WIDTH
C                   PARM           DEPTH
C                   PARM           BYTES
C                   PARM           IDTA
C                   Z-ADD50        XMOV
C                   Z-ADD50        YMOV
C                   CALL 'GDDM'                          Move to
C                   PARM 'GSMOVE'  GSMOVE  8             position
C                   PARM           XMOV
C                   PARM           YMOV
C                   CALL 'GDDM'                          Draw scaled
C                   PARM 'GSIMGS'  GSIMGS  8             image
C                   PARM           ITYPE
C                   PARM           WIDTH
C                   PARM           DEPTH
```

```
     C                   PARM           BYTES
     C                   PARM           IDTA
     C                   PARM           SCALX
     C                   PARM           SCALY
     C                   CALL 'GDDM'                          Send images
     C                   PARM 'ASREAD'  ASREAD  8             to display
     C                   PARM           ATTYPE
     C                   PARM           ATTVAL
     C                   PARM           COUNT
     C                   CALL 'GDDM'                          Terminate
     C                   PARM 'FSTERM'  FSTERM  8             graphics
     C                   SETON                      LR
     C                   RETRN
**
THE FOLLOWING IMAGES WERE DONE IN RPG
**
USING THE BITON AND BITOF COMMANDS
```

Graphics Image Drawn in the COBOL/400 Programming Language




```
-A+++B++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CIMAGE.
*---------------------------------------------------------------
*  This COBOL program draws an image and a scaled image.
*  A BASIC program is called to convert the string to a
*  bit pattern.
*---------------------------------------------------------------
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.   IBM-S38.
 OBJECT-COMPUTER.   IBM-S38.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
 DATA DIVISION.
 FILE SECTION.
 WORKING-STORAGE SECTION.
 01  PARM-LIST.
     05  IMAGE                  PIC X(248).
*****************************************************************
*   Parameters for GDDM routines.
*****************************************************************
 77  FKTYPE PIC S9(5) COMP-4.
 77  FKNUM PIC S9(5) COMP-4.
 77  NUM PIC S9(5) COMP-4.
 77  STRING-LENGTH PIC S9(5) COMP-4.
 77  XMOV PIC S9(4)V9 COMP-3.
 77  YMOV PIC S9(4)V9 COMP-3.
 77  SCALX PIC S9(4)V99 COMP-3.
 77  SCALY PIC S9(4)V99 COMP-3.
 77  WIDTH PIC S9(5) COMP-4.
```

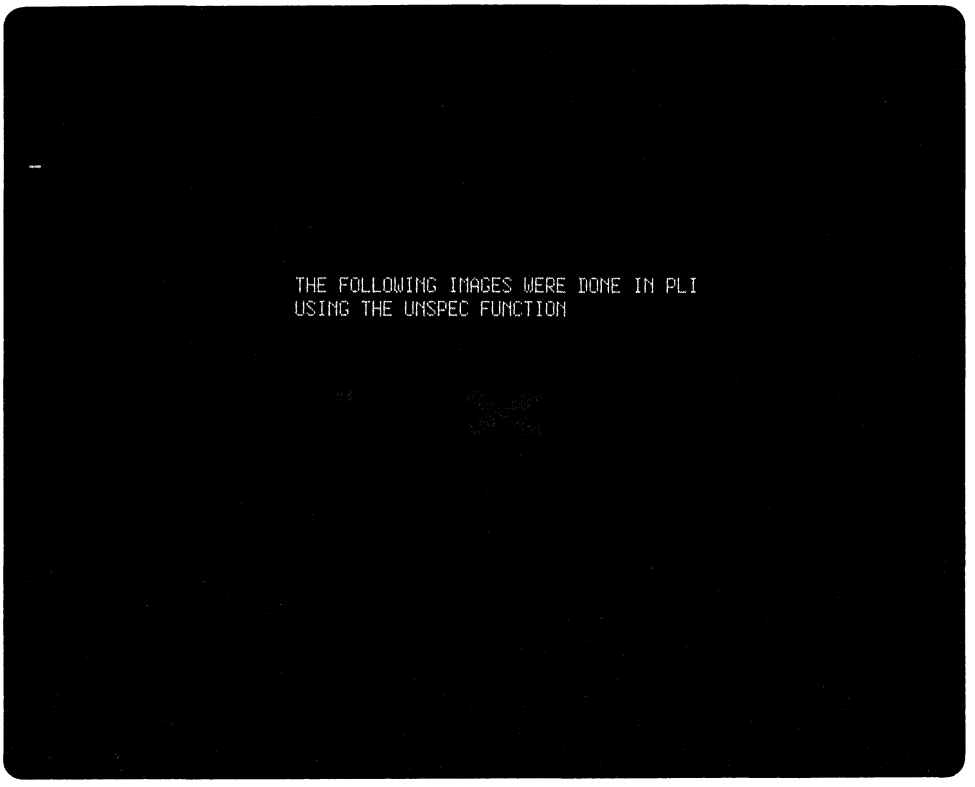
```
 77  DEPTH PIC S9(5) COMP-4.
 77  BYTES PIC S9(5) COMP-4.
 77  ITYPE PIC S9(5) COMP-4.
 77  COLOR PIC S9(5) COMP-4.
 77  STRNG PIC X(50).
*****************************************************************
*    GDDM routines.
*****************************************************************
 77  FSINIT PIC X(8) VALUE "FSINIT  ".
 77  GSIMG  PIC X(8) VALUE "GSIMG   ".
 77  GSIMGS PIC X(8) VALUE "GSIMGS  ".
 77  GSCOL  PIC X(8) VALUE "GSCOL   ".
 77  GSMOVE PIC X(8) VALUE "GSMOVE  ".
 77  GSCHAP PIC X(8) VALUE "GSCHAP  ".
 77  ASREAD PIC X(8) VALUE "ASREAD  ".
 77  FSTERM PIC X(8) VALUE "FSTERM  ".
 PROCEDURE DIVISION.
 GRAPHICS.
*****************************************************************
*  Convert the character string of 1's and 0's that represent
*  the image in bits into an actual bit string. BASCNVT is
*  a BASIC program that performs the conversion.
*****************************************************************
     MOVE "11000011011001100001100001100110110000011" TO IMAGE.
     CALL "BASCNVT" USING IMAGE.
*****************************************************************
*  Initialize graphics.
*****************************************************************
     CALL "GDDM" USING FSINIT.
*****************************************************************
*  Write the heading characters.
*****************************************************************
     MOVE 2 TO COLOR.
     CALL "GDDM" USING GSCOL, COLOR.
     MOVE 30 TO XMOV.
     MOVE 70 TO YMOV.
     CALL "GDDM" USING GSMOVE, XMOV, YMOV.
     MOVE 39 TO STRING-LENGTH.
     MOVE "THE FOLLOWING IMAGES WERE DONE IN COBOL" TO STRNG.
     CALL "GDDM" USING GSCHAP, STRING-LENGTH, STRNG.
     MOVE 30 TO XMOV.
     MOVE 65 TO YMOV.
     MOVE 43 TO STRING-LENGTH.
     MOVE "USING BASIC TO DO THE BIT STRING CONVERSION" TO STRNG.
     CALL "GDDM" USING GSMOVE, XMOV, YMOV.
     CALL "GDDM" USING GSCHAP, STRING-LENGTH, STRNG.
*****************************************************************
*  Draw the images.
*****************************************************************
     MOVE 1 TO COLOR.
     CALL "GDDM" USING GSCOL, COLOR.
     MOVE 35.0 TO XMOV.
     MOVE 50.0 TO YMOV.
     CALL "GDDM" USING GSMOVE, XMOV, YMOV.
     MOVE 0 TO ITYPE.
     MOVE 8 TO WIDTH.
     MOVE 5 TO DEPTH.
     MOVE 5 TO BYTES.
     CALL "GDDM" USING GSIMG, ITYPE, WIDTH, DEPTH, BYTES, IMAGE.
     MOVE 10 TO SCALX.
```

```
           MOVE 10 TO SCALY.
           MOVE 50.0 TO XMOV.
           MOVE 50.0 TO YMOV.
           CALL "GDDM" USING GSMOVE, XMOV, YMOV.
           CALL "GDDM" USING GSIMGS, ITYPE, WIDTH, DEPTH, BYTES, IMAGE,
                 SCALX, SCALY.
      ************************************************************
      *  Display the images and terminate graphics.
      ************************************************************
           CALL "GDDM" USING ASREAD, FKTYPE, FKNUM, NUM.
           CALL "GDDM" USING FSTERM.
           STOP RUN.
         END-GRAPHICS.
```

The following BASIC program converts the bit string to the character string needed by the GSIMG and GSIMGS routines in the COBOL/400 program:

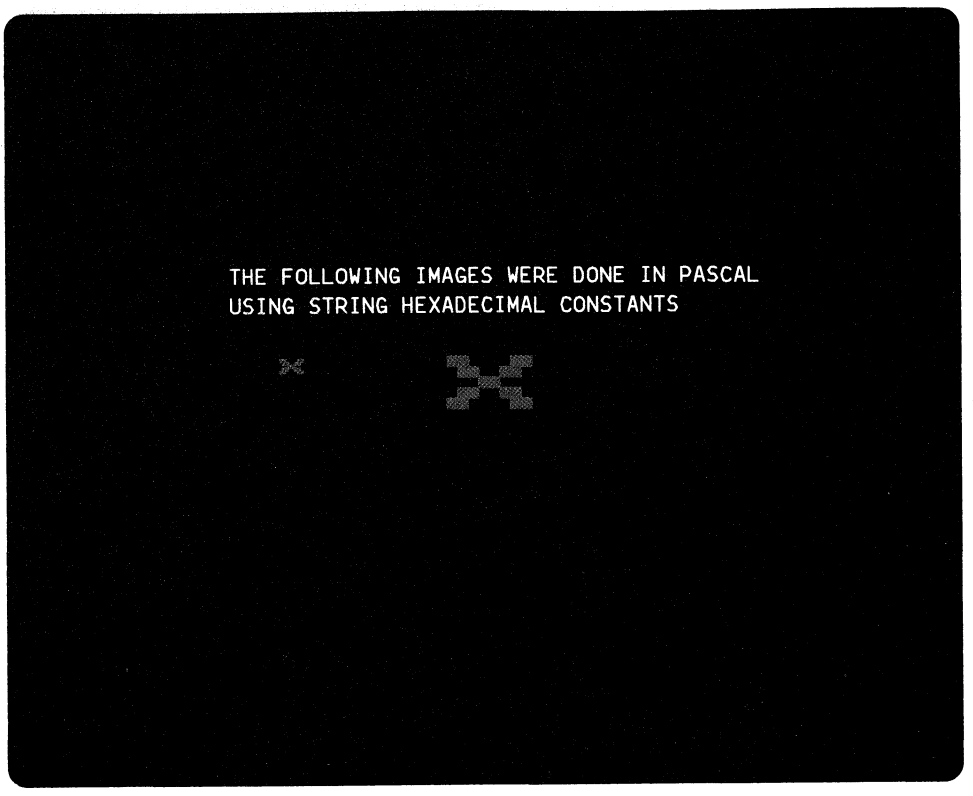
```
00100 SUB BASCNVT (BITIMG$)
00110 !   This program converts the string of 1's and 0's
00120 !   passed by the COBOL program into a character string
00140 DECLARE PROGRAM BASCNVT (C 248)
00150 DIM BITIMG$*248
00160 DIM BITS$*4
00170 DIM HEXS$*1
00180 DIM HEXIMG$*248
00185 GOSUB CNVRT
00190 BITIMG$=HEX$(HEXIMG$)
00200 STOP
00210 CNVRT:     ! CONVERTS A BIT STRING TO HEX
00220  LENG = LEN(BITIMG$)
00230  X= LENG / 4
00240  J = 1
00250  K = 4
00260  FOR I = 1 TO X
00270    BITS$ = BITIMG$(J:K)
00280    J = J+4
00290    IF BITS$(1:1) = '1' THEN GOSUB GRTEQ8 ELSE GOSUB LESS8
00300    K = K+4
00310    HEXIMG$ = HEXIMG$&HEXS$
00320  NEXT I
00330 RETURN
00340 GRTEQ8:  ! HEX IS GREATER THAN OR EQUAL 8
00350    IF BITS$(1:4) = '1000' THEN HEXS$ = '8'
00360    IF BITS$(1:4) = '1001' THEN HEXS$ = '9'
00370    IF BITS$(1:4) = '1010' THEN HEXS$ = 'A'
00380    IF BITS$(1:4) = '1011' THEN HEXS$ = 'B'
00390    IF BITS$(1:4) = '1100' THEN HEXS$ = 'C'
00400    IF BITS$(1:4) = '1101' THEN HEXS$ = 'D'
00410    IF BITS$(1:4) = '1110' THEN HEXS$ = 'E'
00420    IF BITS$(1:4) = '1111' THEN HEXS$ = 'F'
00430 RETURN
00440 LESS8: ! HEX IS LESS THAN 8
00450   IF BITS$(1:4) = '0000' THEN HEXS$ = '0'
00460   IF BITS$(1:4) = '0001' THEN HEXS$ = '1'
00470   IF BITS$(1:4) = '0010' THEN HEXS$ = '2'
00480   IF BITS$(1:4) = '0011' THEN HEXS$ = '3'
00490   IF BITS$(1:4) = '0100' THEN HEXS$ = '4'
00500   IF BITS$(1:4) = '0101' THEN HEXS$ = '5'
00510   IF BITS$(1:4) = '0110' THEN HEXS$ = '6'
00520   IF BITS$(1:4) = '0111' THEN HEXS$ = '7'
00530 RETURN
00540 END SUB
```

Graphics Image Drawn in PL/I




```
IMAGEPLI:PROC;
 %INCLUDE SYSLIB (ADMUPLNB);
DCL ATTYPE BIN(31);                  /* Declare variables       */
DCL ATTVAL BIN(31);
DCL COUNT BIN(31);
DCL IMAGE$ CHAR(5);
/* The following bit string represents the image data           */
UNSPEC(IMAGE$) = '11000011'B||
                 '01100110'B||
                 '00011000'B||
                 '01100110'B||
                 '11000011'B;
CALL FSINIT;                         /* Initialize graphics     */
CALL GSCOL(2);                       /* Set color               */
CALL GSCHAR(30.0,70.0,37,'THE FOLLOWING IMAGES WERE DONE IN PLI');
CALL GSCHAR(30.0,65.0,37,'USING THE UNSPEC FUNCTION           ');
CALL GSCOL(1);                       /* Set color               */
CALL GSMOVE(35.0,50.0);              /* Move to image position  */
CALL GSIMG(0,8,5,5,IMAGE$);          /* Draw image              */
CALL GSMOVE(50.0,50.0);              /* Move to image position  */
CALL GSIMGS(0,8,5,5,IMAGE$,10.0,10.0);/* Draw scaled image      */
CALL ASREAD(ATTYPE,ATTVAL,COUNT);    /* Send picture to display */
END IMAGEPLI;
```

Graphics Image Drawn in Pascal




```
PROGRAM IMAGEPAS;

TYPE
  %INCLUDE QATTPAS(ADMUSTNO);       /* IBM-supplied type declarations */

VAR
  IMAGE$ : CHARARR_2040;
  ATTVAL, ATTYPE, COUNT : INTEGER;
  A, B, C, D : INTEGER;
  X, Y, Z : SHORTREAL;
  CHARSTRING : CHARARR_132;

%INCLUDE QATTPAS(ADMUSLNB);         /* IBM-supplied proc declarations */

BEGIN
  IMAGE$ := 'C3661866C3'XC;         /* The following hexidecimal
                                       character string represents the
                                       image data                    */
  FSINIT;                           /* Initialize graphics           */
  A := 2;
  GSCOL(A);                         /* Set color                     */
  X := 30.0; Y := 70.0; A := 40;
  CHARSTRING := 'THE FOLLOWING IMAGES WERE DONE IN PASCAL';
  GSCHAR(X,Y,A,CHARSTRING);         /* Write first line of text      */
  Y := 65.0;
  CHARSTRING := 'USING STRING HEXIDECIMAL CONSTANTS      ';
  GSCHAR(X,Y,A,CHARSTRING);         /* Write second line of text     */
  A := 1;
  GSCOL(a);                         /* Set color                     */
  X := 35.0; Y := 50.0;
  GSMOVE(X,Y);                      /* Move to image position        */
```

```
  A := 0; B := 8; C := 5; D := 5;
  GSIMG(A,B,C,D,IMAGE$);            /* Draw image                      */
  X := 50.0;
  GSMOVE(X,Y);                      /* Move to scaled image position  */
  X := 10.0; Y := 10.0;
  GSIMGS(A,B,C,D,IMAGE$,X,Y);       /* Draw scaled image              */
  ASREAD(ATTVAL,ATTYPE,COUNT)       /* Send picture to display        */
END.
```

Appendix A. Devices Compatible with the AS/400 System

An OS/400 graphics program can produce these types of output:

Video color pictures on the following devices:

- IBM personal computer with work station function (WSF)
- IBM personal computer with work station emulation (WSE)
- 5292 Model 2
- IBM personal computer with 5250 emulation

Plotted color pictures on the following devices:

- IBM 6180 Plotter
- IBM 6182 Plotter
- IBM 6184 Plotter
- IBM 6185 Plotter
- IBM 6186-1 Plotter
- IBM 6186-2 Plotter
- IBM 7371 Plotter
- IBM 7372 Plotter

Printed black-and-white pictures on the following devices:

- IBM 3812 Printer
- IBM 3816 Printer
- IBM 4028 Printer
- IBM 4214 Printer
- IBM 4234-2 Printer
- IBM 5224 Printer
- IBM 5225 Printer

Printed color pictures on the IBM 4224 Printer.

Graphics data format (GDF) file, which can be used to save a constructed picture for later interpretation, on the AS/400 System or on another system equipped with the software necessary to interpret GDF files.

No output on a non-graphics work station, which can be useful for testing and debugging programs when a graphics work station is not available.

A plotter can be attached to the graphics work station as an auxiliary device, and therefore must be named and addressed in the AUXDEV parameter of the CRTDEVDSP (Create Device Description (Display)) CL command used to describe the graphics work station to the system.

The 3812, 3816, 4028, 4214, 4224, 4234-2, 5224, and 5225 printers are separate devices and are configured as normal work station printers.

The same picture produced on a display, a plotter, and a printer may not be identical. Pictures are associated with a device. When the program encounters a DSUSE routine for the plotter, the picture constructed differs from the graphics work station picture:

- Chart features, such as the legend and chart notes, are positioned by character grid units. The default character grid varies between the display device, printer, and plotter. On a plotter, paper sizes and paper orientation make a difference. To avoid these differences, set the character grid size with Presentation Graphics routine CHCGRD.

© Copyright IBM Corp. 1991

- Charts with default axis scales show more tick marks on a plotted or printed version of a chart than on one displayed.
- Note, legend, and axis label *blanking* is not performed for plotted charts unlike displayed or printed charts.
- Color tables are not supported for the plotter; regardless of the color defined for the color table entry number 1 for the display, the plotter uses whatever pen is in position 1 of the plotter carousel.
- Color mixing for plotter pictures is always in *overpaint* mode, because colors are etched over the top of previously drawn colors.
- Area-fill patterns are shown in much finer detail on the plotter and printer than on the display.
- The default character mode for printers is mode-2 (image symbols), while for plotters and displays the default character mode is mode-3 (vector symbols).
- The plotter and printer do not reproduce alphanumeric display file fields written over the picture.

For screen copies produced by the 5182 Color Printer, or print files produced for a graphics work station printer:

Color mixing for printer pictures is always in *overpaint* mode, because colors are etched over the top of previously drawn colors.

The printer does not produce the colors defined in modified device color tables. It uses the color table indexes and prints the colors accordingly.

The printer does not reproduce display file field attributes. For example, alphanumeric fields entered into input fields displayed with the underline attribute will be reproduced but *not* with the underline.

When you call a compiled high-level language graphics program (or a BASIC source program from the BASIC interpreter), the device enters *graphics display mode*:

The display is set to reduced line spacing, which cannot be overridden.

The graphics mode indicator (a blue G character two inches from the left side of the screen) is shown.

The graphics work station must be described to the system by issuing the CRTDEVDSP (Create Device Description (Display)) command. After being described, it must be varied online.

For more information, refer to the *CL Reference* manual and the *Licensed Programs and New Release Installation Guide*.

The graphics work station from which you start the program is always the default *current device*. Therefore, if no DSOPEN/DSUSE routines exist in the program to send the program output elsewhere, the picture will be displayed on the graphics work station.

The IBM Plotters

A plotter can be assigned in the program as the current device. The DSOPEN routine identifies the plotter to the program, and the DSUSE routine makes it the current device. (An ASREAD or FSFRCE routine will then send the current page of graphics to the plotter.)

How to Configure a Plotter

A plotter must be described to the system by including it as an auxiliary device to the associated graphics work station. Name the plotter in the AUXDEV parameter of the CRTDEVDSP (Create Device Description (Display)) command used to describe the graphics work station to the system. For more information, refer to the *CL Reference* manual and the *Licensed Programs and New Release Installation Guide*.

How to Send Pictures to a Plotter

To use a plotter, you must use the DSOPEN routine to identify the device and its characteristics to the program.

The DSOPEN routine has these parameters:

- Device identifier. A positive integer greater than 1 that identifies the plotter and is used in the DSUSE routine to make the plotter the current device.
- Family. A positive integer (which must always be 1).
- Device token. An 8-byte character string that identifies the device type. The following are valid device tokens for IBM plotters (each b is one blank space):
  - '6180bbbb'
  - '6182bbbb'
  - '6184bbbb'
  - '6185bbbb'
  - '6186M1bb'
  - '6186M2bb'
  - '7371bbbb'
  - '7372bbbb'
- Processing options list count. A positive integer that specifies the number of processing options you want to specify explicitly in the processing options list (described next).
- Processing options list. An integer array that holds option codes and values. These specify the speed and width of the plotter pens, the size and orientation of the paper in the plotter, and whether or not form feed is enabled.

  The list takes this form:

  Option code, Value, Option code, Value, ...

  The option group codes and the available options are as follows.

  Valid values for each processing option group and plotter are:

Plotter	Processing Option Group					Notes
	10	11	12	15	16	
6180	NV	0 – 100	0 – 10	0 – 2	0 – 2	3
6182	0 – 2	0 – 100	0 – 10	NV	0 – 2	2
6184	NV	0 – 100	0 – 10	NV	0 – 2	
6185	NV	0 – 100	0 – 10	NV	0 – 2	
6186-1	NV	0 – 100	0 – 10	0 – 5	0 – 2	3
6186-2	0 – 2	0 – 100	0 – 10	0 – 5	0 – 2	1, 3
7371	NV	0 – 100	0 – 10	NV	NV	
7372	NV	0 – 100	0 – 10	0 – 2	0 – 2	3

NV - Processing option group is not valid for this plotter.

Notes:

1. Form feed option enables roll feed.
2. Form feed option enables 8.5 x 11 inch sheet feed.
3. Value shown for option group 15 is paper size code. The dimension type code value is 0 – 2.

Option group 10: form feed

Options: An integer of 0 through 2 that specifies whether form feed is enabled or disabled.

Option 1 specifies that form feed is disabled. This means that if FSFRCE or ASREAD call is executed, a page eject does not occur, and a new sheet of paper is not loaded. Option 2 specifies that form feed is enabled. This means that if FSFRCE or ASREAD call is executed, the sheet of paper currently being plotted on is ejected, and a new sheet of paper is loaded in the plotter. The default is 0, which is the same as option 2.

Option group 11: pen speed

Options: Any integer value of 0 through 100; a value of 50 specifies that the pen will move across the paper 50% as fast as maximum speed. 0 is the default, which is the same as maximum speed (100); 1 is the slowest speed possible.

For paper plots, 50 is a good speed. For transparencies, a slower speed (such as 30) should be used to prevent lightly-colored lines.

Option group 12: pen width

Options: Any integer value of 0 through 10, where the value specifies the width (in millimeters) of the pens installed in the plotter.

The pen width you specify determines line spacing of patterns for filled areas. If you specify 10 (1.0 millimeter width), the plotter leaves extra space between lines in patterns; if you specify 1 (.10 millimeter width), the plotter uses finely-spaced lines. The default pen width is 3 (0.3 millimeters).

Option group 15: paper size

Options: Two options can be specified together:

- An integer of 0 through 5 that specifies the paper size code. The default is 0 (same as 1).
  - 1 specifies 8.5 by 11-inch paper
  - 2 specifies 11 by 17-inch paper

- 3 specifies 17 by 22-inch paper
- 4 specifies 22 by 34-inch paper
- 5 specifies 34 by 44-inch paper

— An integer of 0 through 2 that specifies the dimension type code. The dimension type code further qualifies the paper size, in terms of a measurement in millimeters (dimension type code 1) or inches (dimension type code 2). The default is 0, which is the same as type code 1.

The paper size switches should be adjusted on the plotter to match the paper size specified; otherwise, undesirable results can occur. (A picture to be plotted on an 11 by 17 sheet may exceed the boundaries of an 8.5 by 11 sheet.)

Option group 16: paper orientation

Options: An integer of 0 through 2 that specifies whether the picture is plotted vertically on the paper or horizontally.

A picture plotted horizontally (the default 0 or option 1) will be plotted such that the x range is plotted in the longest dimension of the paper (11 for 8.5 by 11 sheets). Option 2 specifies a vertical orientation. The vertical orientation can change the default aspect ratio of the picture (making it look taller), unless a picture space is specified explicitly.

- Name list count. A positive integer that specifies the number of names (always 0, 1, or 2) in the name list (described next).
- Name list. A character array that specifies the OS/400 device description name of the device.

  If a name is specified for the plotter, use the name in the CRTDEVDSP command to describe the associated graphics work station to the system.

  If the name list count is 2, the plotter device is selected based on either of the following entries in the second array element:

' nn'	The plotter (of the type specified by the device token) at address nn is used. The address assigned to the plotter in the device description for the associated work station is represented in this element by two right-justified characters.
'ADMPLOT '	The plotter (of the type specified by the device token) at the lowest address of an available plotter is used.

The DSOPEN parameter must be followed by a DSUSE routine that uses the integer device identifier specified by the first parameter of the DSOPEN routine.

DSUSE makes the plotter the current device.

The following is an example of the DSOPEN and DSUSE routines used to activate the IBM 6180 Plotter:

```
00040 OPTION BASE 1                     ! Set array subscript base
00050 ! ************** Device routines ***************************
00060 INTEGER PLST                      ! Declare integer
00070 DIM PLST(4) : MAT READ PLST       ! Dimension, read array
00080 DATA 11,50,16,1
00090 DIM NLST$(1) : NLST$(1) = ' '     ! Dimension, assign value
00100 CALL GDDM ('DSOPEN',2,1,'6180    ',4,PLST(),0,NLST$())
00110 ! Open plotter device 2 of family 1 named 6180,
00120 ! using PLST option group 11 value 50 (pen speed 50% of max),
00130 ! and using group 16 option 1 (horizontal paper orientation);
00140 ! name list has 0 names in array NLST$
```

```
00150 CALL GDDM ('DSUSE',1,2)
00160 ! Use device 2 as active device (option 1)
00170 ! ***************** Symbol set ****************************
00180 CALL GDDM ('GSLSS',2,'ADMUWCIP',66)
00190       .
00190       .
00190       .
00200   (GDDM or Presentation Graphics program follows)
```

Printers Capable of Graphics

The following AS/400 printers are capable of graphics:

3812 Printer
3816 Printer
4028 Printer
4214 Printer
4224 Printer
4234-2 Printer
5224 Printer
5225 Printer

These printers can print graphics directly from the application program (SPOOL(*NO) output or output that is not spooled), or they can print previously spooled graphics files from an output queue. In either case, a graphics printer file must be opened. For more information on graphics printer files, refer to Chapter 5, "OS/400 Programming Considerations." Opening graphics printer files is described below.

How to Configure a Printer

Printers that are capable of graphics, are configured the same as any other work station printer. For more information, refer to the *Licensed Programs and New Release Installation Guide*.

How to Send Pictures to a Printer

Graphics printer files are opened when the DSOPEN routine is run. The DSOPEN routine identifies the printer file to be used (that is, whether the output is spooled or not) with these parameters:

Device identifier. A positive integer greater than 1 that identifies the printer and is used in the DSUSE routine to make the printer the current device.

Family. A positive integer (which must always be 1).

Device token. An 8-byte character string that identifies the device type. The following device tokens are associated with these work station printers:

Token	Device
522X	5224 and 5225 printers
4234	4234-2 printer
4214	4214 printer
IPDS	3812, 3816, 4028, and 4224 printers

For example, to open a file to the 4214 Printer, use the string '4214bbbb' (each b is one blank space).

Name list count. A positive integer that specifies the number of names (always 0 or 1) on the name list.

Name list. A single-element character array that specifies the graphics printer file to be used. If no graphics printer file name is entered in the name list while

a graphics printer device token is specified, the default printer file QPGDDM is used (with any overrides in effect for it).

This is an example of the DSOPEN and DSUSE routines used to open a 5224 or 5225 Printer:

```
00010 CALL GDDM ('FSINIT')              ! Initialize graphics
00020 INTEGER DEVID,FAM,PCT,PLST,NCT,US ! Declare integers
00030 DEVID=2 : FAM=1 : PCT=0           ! Variables for printer DSOPEN
00040 ! Device #2, family type 1, 0 items in device parameter list
00050 DIM PLST(0)                       ! Declare parameter list
00060 NCT = 0                           ! Name list is empty
00070 CALL GDDM ('DSOPEN',DEVID,FAM,'522X    ',PCT,PLST(),NCT,' ')
00080 ! Open device #2, family type 1, 5224 or 5225 printer,
00090 ! name count 0, use default printer file (name-list ' ')
00100 US=1                             ! Variable for printer DSUSE
00110 CALL GDDM ('DSUSE',US,DEVID)     ! Use device #2 as current device
00120      .
00130      .
00140   GDDM or Presentation Graphics program
00150      .
00160      .
00170 CALL GDDM ('FSFRCE')             ! Close printer file
00180 ! *************** END GRAPHICS ****************************
00190 CALL GDDM ('FSTERM')             ! End graphics
```

Here is an example of a program that uses a user-defined file as a graphics printer file:

First, the file is created:

```
CRTPRTF FILE(QTEMP/PRINT) PAGESIZE(99 132) LPI(9) CPI(15)
```

Then, the program opens the user-defined file by including its name on the name-list parameter of DSOPEN:

```
00010 CALL GDDM ('FSINIT')              ! Initialize graphics
00020 INTEGER DEVID,FAM,PCT,PLST,NCT,US ! Declare integers
00030 DEVID=2 : FAM=1 : PCT=0           ! Variables for printer DSOPEN
00040 ! Device #2, family type 1, 0 items in device parameter list
00050 DIM PLST(0)                       ! Declare parameter list
00060 NCT = 1                           ! Name list has one name
00070 CALL GDDM ('DSOPEN',DEVID,FAM,'4214    ',PCT,PLST(),NCT,'PRINT')
00080 ! Open device #2, family type 1, 4214 Printer,
00090 ! name count 1, open user-defined printer file PRINT
00100 US=1                             ! Variable for printer DSUSE
00110 CALL GDDM ('DSUSE',US,DEVID)     ! Use device #2 as current device
00120      .
00130      .
00140   GDDM or Presentation Graphics program
00150      .
00160      .
00170 CALL GDDM ('FSFRCE')             ! Close printer file
00180 ! *************** END GRAPHICS ****************************
00190 CALL GDDM ('FSTERM')             ! End graphics
```

Merging Text and Graphics for Print Files

You can merge text and graphics for print files. For the 5224, 5225, 4214, and 4234-2 printers, merging text and graphics is only supported for program-described print files and the application must be coded in the RPG/400 programming language.

For IPDS printers, merging text and graphics is supported for both program-described print files and externally described print files. For program described print files, the application program must be coded in the RPG/400 programming language. For externally described print files, the application program can be in BASIC, the COBOL/400 language, Pascal, PL/I, or the RPG/400 programming language.

For both program-described and externally described print files, there are some rules that must be followed to merge text and graphics:

1. Open the print file with SHARE(*YES). Specify SHARE(*YES) on the OVRPRTF command. If this is not done, two separate print files (one for text, and one for graphics) are produced.

   The name of the print file specified on the GDDM DSOPEN call statement must be the same as the name of the print file specified on the RPG OPEN statement when opening the print file for text. In the example below, both are called QPGDDM.

2. The print file must be opened for graphics before it can be opened for text. Specify the GDDM DSOPEN statement first before you open the print file with an OPEN statement.

3. PUTs for text may be intermixed with PUTs for graphics. When printing graphics on a page, use the GDDM FSFRCE statement to print the page. The GDDM FSFRCE statement will cause any graphics created for the page (and also text) to be printed. The GDDM FSFRCE statement also causes a page eject and skips to line one of the next page.

   If a page contains text and graphics, you must use the GDDM FSFRCE to eject the page. Do not skip to a new page with controls specified from the program-described or externally described file.

   If a page contains only text (no graphics), do not use the GDDM FSFRCE statement. Use print controls for program-described or externally described print files to skip to a new page.

4. There is no required order for closing the print file. The file is closed for graphics by using the GDDM FSTERM statement, and for text by using the CLOSE statement of the language you are using.

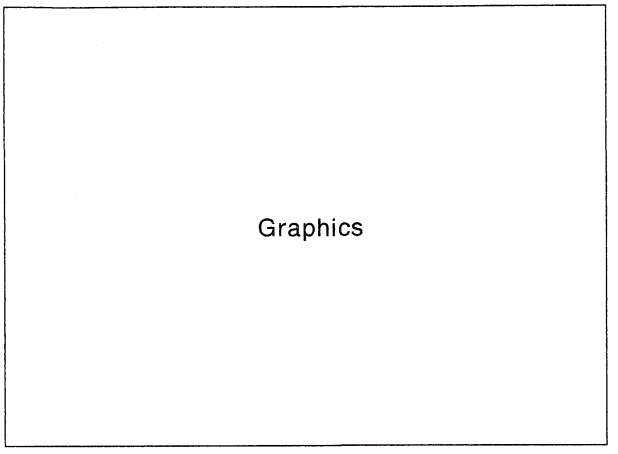
This is an example of a two-column page that has text in one column and a graphics drawing in the other column. The RPG statements that produced the output are shown following the output.

It is possible to put text on one side of the page and graphics on the other.

The program that generated this output was written in the RPG/400 programming language.

Some things to remember when writing such an RPG/400 program are:

1) The print file must be opened SHARE(*YES). Specify SHARE(*YES) on the OVRPRTF command.
2) The print file must be opened FIRST (with DSOPEN).
3) Define the print file as having program control so that it can be explicitly opened by the program and not opened implicitly by the RPG/400 programming language.
4) The name of the print file specified on the DSOPEN must be the same as the file specified on the RPG OPEN statement.




```
0000000001111111111222222222233333333334444444444555555555566666666667777777
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
     FQPGDDM  O   F     132            PRINTER                        UC
     E                    ARR1    1   4  9 0
     E                    ARR2        1 10
     E                    ARR3    1  33 80
     IDS           DS
     I                                    B   1   40LEN
     I                                    B   5   80X
     I                                    B   9  120Y
     I                                    B  13  160Z
     I                                    B  17  200V
     I                                    B  21  360ARR1
     I                                    B  37  400USG
     I                                    B  41  440DID
     C                     CALL 'GDDM'
     C                     PARM 'FSINIT'  FSINIT  8
     C                     CALL 'GDDM'
     C                     PARM 'DSOPEN'  DSOPEN  8
     C                     PARM 2         X
     C                     PARM 1         Y
     C                     PARM '522X    'DEV     8
     C                     PARM 0         Z
     C                     PARM           ARR1
     C                     PARM 1         V
     C                     PARM 'QPGDDM'  ARR2,1
     C                     CALL 'GDDM'
     C                     PARM 'DSUSE'   DSUSE   8
     C                     PARM 1         USG
     C                     PARM 2         DID
     C                     CALL 'GDDM'
     C                     PARM 'GSCHAR'  GSCHAR  8
     C                     PARM 80        X1      51
     C                     PARM 85        Y1      51
```

```
C                   PARM 8          LEN
C                   PARM 'Graphics'STR      8
C                   CALL 'GDDM'
C                   PARM 'GSMOVE'   GSMOVE  8
C                   PARM 70         X2      51
C                   PARM 70         Y2      51
C                   CALL 'GDDM'
C                   PARM 'GSLINE'   GSLINE  8
C                   PARM 100        X2
C                   PARM 70         Y2
C                   CALL 'GDDM'
C                   PARM 'GSLINE'   GSLINE  8
C                   PARM 100        X2
C                   PARM 100        Y2
C                   CALL 'GDDM'
C                   PARM 'GSLINE'   GSLINE  8
C                   PARM 70         X2
C                   PARM 100        Y2
C                   CALL 'GDDM'
C                   PARM 'GSLINE'   GSLINE  8
C                   PARM 70         X2
C                   PARM 70         Y2
C                   OPEN QPGDDM
C                   MOVEAARR3,1     DATA   80
C                   EXCPT
C                   MOVEAARR3,2     DATA
C                   EXCPT
C                   MOVEAARR3,3     DATA
C                   EXCPT
C                   MOVEAARR3,4     DATA
C                   EXCPT
C                   MOVEAARR3,5     DATA
C                   EXCPT
C                   MOVEAARR3,6     DATA
C                   EXCPT
C                   MOVEAARR3,7     DATA
C                   EXCPT
C                   MOVEAARR3,8     DATA
C                   EXCPT
C                   MOVEAARR3,9     DATA
C                   EXCPT
C                   MOVEAARR3,10    DATA
C                   EXCPT
C                   MOVEAARR3,11    DATA
C                   EXCPT
C                   MOVEAARR3,12    DATA
C                   EXCPT
C                   MOVEAARR3,13    DATA
C                   EXCPT
C                   MOVEAARR3,14    DATA
C                   EXCPT
C                   MOVEAARR3,15    DATA
C                   EXCPT
C                   MOVEAARR3,16    DATA
C                   EXCPT
C                   MOVEAARR3,17    DATA
C                   EXCPT
C                   MOVEAARR3,18    DATA
C                   EXCPT
C                   MOVEAARR3,19    DATA
```

```
     C                   EXCPT
     C                   MOVEAARR3,20   DATA
     C                   EXCPT
     C                   CALL 'GDDM'
     C                   PARM 'FSFRCE'  FSFRCE  8
     C                   CALL 'GDDM'
     C                   PARM 'FSTERM'  FSTERM  8
     C                   SETON                     LR
     OQPGDDM  E 1
     O                       DATA     100
**
000000000
000000000
000000000
000000000
**
```

It is possible to put text on one side of
the page and graphics on the other.

The program that generated this output was
written in the RPG/400 programming language.

Some things to remember when writing
such an RPG/400 program are:

1) The print file must be opened SHARE(*YES). Specify
   SHARE(*YES) on the OVRPRTF command.
2) The print file must be opened FIRST (with DSOPEN).
3) Define the print file as having program control
   so that it can be explicitly opened by the
   program and not opened implicitly by the RPG/400
   programming language.
4) The name of the print file specified on the DSOPEN must
   be the same as the file specified on the RPG OPEN statement.

Non-Graphics Devices

You can use a non-graphics device to call graphics programs (or interpreted BASIC source files). When you invoke such a program from a non-graphics device, the program executes normally, but no picture is displayed and no values are returned to device query routines in the program. Display file fields, however, *are* sent to the device screen.

You can use a dummy device to test and debug graphics programs, because messages about the program execution are sent to the device message queue.

A dummy device *must* be used when a program is executed to retrieve a graphics data format (GDF) file. You can simulate a dummy device with a graphics work station by using a blank name (' ') in the name list parameter of the DSOPEN routine.

Bibliography

The following AS/400 manuals contain information you may need. The manuals are listed with their full title and base order number. When these manuals are referred to in this manual, the short title is used.

- *Application Development Tools: Source Entry Utility User's Guide and Reference*, SC09-1338 provides the application programmer or programmer with information about using the Application Development Tools source entry utility (SEU) to create and edit source members. The manual explains how to start and end an SEU session and how to use the many features of this full-screen text editor. The manual contains examples to help both new and experienced users accomplish various editing tasks, from the simplest line commands to using pre-defined prompts for high-level languages and data formats.

  Short Title: *SEU User's Guide and Reference*

- *Business Graphics Utility User's Guide and Reference*, SC09-1408 provides the application programmer, programmer, system administrator, or business or technical professional with information about using the AS/400 Business Graphics Utility (BGU) to create various types of charts. It is divided into two sections: the first section includes several exercises that familiarize the user with the functions of BGU, and the second section contains reference material.

  Short Title: *BGU User's Guide and Reference*

- *Data Description Specifications Reference*, SC41-9620, provides the application programmer with detailed descriptions of the entries and keywords needed to describe database files (both logical and physical) and certain device files (for displays, printers, and ICF) external to the user's programs.

  Short Title: *DDS Reference*

- *Languages: BASIC User's Guide and Reference*, SC09-1157, provides the application programmer with the information needed to write, test, and maintain AS/400 BASIC programs in both the AS/400 environment and the System/38 environment.

  Short Title: *BASIC User's Guide and Reference*

- *Languages: Pascal User's Guide*, SC09-1209, provides the application programmer with information about how to use the AS/400 Pascal compiler. The manual explains how to enter, compile, run, and debug AS/400 Pascal programs. It also describes how to use input and output (I/O) function and storage.

  Short Title: *Pascal User's Guide*

- *Languages: PL/I User's Guide and Reference*, SC09-1156, provides the application programmer with information about using AS/400 PL/I in the System/38 environment. Differences between the System/38 environment and the AS/400 environment are identified as well as the enhancements available in the AS/400 environment.

  Short Title: *PL/I User's Guide and Reference*

- *Languages: Systems Application Architecture* AD/Cycle* RPG/400* User's Guide*, SC09-1348, provides the application programmer with the information needed to write, test, and maintain RPG/400 programs on the AS/400 system. The manual provides information on data organizations, data formats, file processing, multiple file processing, automatic report function, RPG command statements, testing and debugging functions, application design techniques, problem analysis, and compiler service information. The differences between the System/38 RPG III, System/38-compatible RPG, and RPG/400 are identified.

  Short Title: *RPG/400* User's Guide*

- *Languages: System/38-Compatible COBOL User's Guide and Reference*, SC09-1159, provides the application programmer with information about using System/38-compatible COBOL on the AS/400 system.

  It provides information on how to program in COBOL on the AS/400 system, like a System/38, and how to use existing System/38 COBOL programs.

  Short Title: *System/38-Compatible COBOL User's Guide and Reference*

- *Programming: Control Language Reference*, SC41-0030, provides the application programmer with a description of the AS/400 control language (CL) and its commands. Each command description includes a syntax diagram, parameters, default values, keywords, and an example. The information should be used to refer to the control language commands to request functions of the Operating System/400 (5728-SS1) licensed program and of the various languages and utilities.

  Short Title: *CL Reference*

- *Programming: GDDM Programming Reference*, SC41-0537, provides the application programmer with information about using OS/400 graphical data display manager (GDDM) to write graphics application programs. This manual provides detailed descriptions of all graphics routines available in GDDM. It also provides information about high-level language interfaces for GDDM.

  Short Title: *GDDM Programming Reference*

© Copyright IBM Corp. 1991

- *Programming: Reference Summary*, SX41-0028 provides the system operator or system administrator with quick information about the organization of the AS/400 commands. This manual contains an alphabetic list of all AS/400 commands and a list, by command, of error messages the programmer can monitor for when writing programs.

  Short Title: *Programming Reference Summary*

Index

A

© Copyright IBM Corp. 1991

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Numerics

Reader's Comments

Application System/400
Programming:
GDDM Programming Guide
Version 2

Publication No. SC41-0536-00

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.

IBM®

Cut
Alor

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN DEPT 245
IBM CORPORATION
3605 HWY 52 N
ROCHESTER MN 55901-7899

Fold and Tape | **Please do not staple** | Fold and Tape

Cut
Alon

Program Number: 5738-SS1

Printed in U.S.A.

SC41-0536-00