ALPHA MICROSYSTEMS
RIGHT. FROM THE START.

# AMIGOS Graphics Operating System Reference Manual

DSM-00058-01 A00

© 1995 Alpha Microsystems

| REVISIONS INCORPORATED | |
|---|---|
| REVISION | DATE |
| 00 | August 1988 |
| 01 | April 1990 |

*AMIGOS Reference Manual*

To re-order this document, request part number DS0-00058-00

The information contained in this manual is believed to be accurate and reliable. However, no responsibility for the accuracy, completeness or use of this information is assumed by Alpha Microsystems.

This document may contain references to products covered under U.S. Patent Number 4,530,048.

The following are registered trademarks of Alpha Microsystems, Santa Ana, CA 92799:

| | | | |
|---|---|---|---|
| AMIGOS | AMOS | Alpha Micro | AlphaACCOUNTING |
| AlphaBASIC | AlphaCALC | AlphaCOBOL | AlphaDDE |
| AlphaFORTRAN 77 | AlphaLAN | AlphaLEDGER | AlphaMAIL |
| AlphaMATE | AlphaNET | AlphaPASCAL | AlphaRJE |
| AlphaWRITE | CASELODE | OmniBASIC | VER-A-TEL |
| VIDEOTRAX | | | |

The following are trademarks of Alpha Microsystems, Santa Ana, CA 92799:

| | | | |
|---|---|---|---|
| AlphaBASIC PLUS | AlphaVUE | AM-PC | AMTEC |
| DART | ESP | MULTI | *in*Sight/am |
| *in*Front/am | | | |

All other copyrights and trademarks are the property of their respective holders.

ALPHA MICROSYSTEMS
2722 S. Fairview St.
P.O. Box 25059
Santa Ana, CA 92799

# TABLE OF CONTENTS

**CHAPTER 4 - THE GRAPHICS CONTROL BLOCK**

**CHAPTER 5 - THE GRAPHICS DEVICE DRIVER**



**CHAPTER 6 - AMIGOS REFERENCE LISTS**



**CHAPTER 7 - REFERENCE SHEETS**

# PREFACE

## ACKNOWLEDGEMENTS

The Alpha Micro Graphics Operating System (AMIGOS) is based on, and contains many of the concepts of, the proposed Computer Graphics Interface and the Graphics Kernal System (GKS). As such, the design of the product is based on the work of many groups.

Much of the early design methodology was developed at the Workshop on Graphics Standards Methodology held in May 1976 in Seillac, France under IFIP WG5.2 sponsership. GKS itself was originally developed by the West German Standardization Institute, DIN, in 1978 and was subsequently refined extensively during the period 1980-1982 by Working Group 2 of the Subcommittee on Programming Languages of the Technical Committee on Information Processing of the International Standard Organization (ISO TC97/SC5/WG2).

The resulting draft International Standard ISO/DIS 7942 then became the draft of the American National Standard for GKS. GKS, in turn, was heavily influenced throughout its development cycle by the work of the Graphics Standards Planning Committee of the Special Interest Group on Computer Graphics of the Association for Computing Machinery (ASM-SIGGRAPH GSPC). This work, known as the CORE SYSTEM proposal, was published and widely distributed in 1977 and again (in a revised version) in 1979.

This manual makes liberal use of the explanations of underlying concepts contained in this earlier work.

## READER AUDIENCE

We assume the reader of this manual is familar with AMOS, and either the Assembler, AlphaC or AlphaBASIC programming languages.

This reference manual is most emphatically **not** a tutorial on using a graphics operating system. However, many explanatory books do exist. Please refer to the section "Reference Books," in Chapter 1 for a list of books you might find useful.

# CHAPTER 1

# INTRODUCTION

This chapter introduces you to AMIGOS, Alpha Microsystems' Graphics Operating System. It discusses the AMIGOS documents available, reference books you might find useful, how this book is organized, and the printing conventions we use in this book.

## 1.1 WHAT IS AMIGOS?

AMIGOS is organized as a collection of subroutines which may be used by an application program to perform graphical input, output, and transformation. AMIGOS provides a standardized interface between the various types of graphical input and output devices (printers, plotters, CRTs, etc.) and application software.

By providing this standard interface, you can write application software so graphics can be displayed on many different device types without modifying the application. With the graphics market in a state of flux, this device independence allows your application to function on many types of devices.

AMIGOS is device independent because it uses a ***graphics device driver*** (GDV) which performs all device dependent translations. The GDV is similar in concept to the terminal driver (TDV) used by the AMOS terminal service system to provide terminal type independence.

## 1.2 THE AMIGOS DOCUMENTATION LIBRARY

The AMIGOS product's documentation library is especially for programmers and consists of these books:

- *AMIGOS Reference Manual* - gives a brief introduction to graphics systems in general and includes detailed information for all AMIGOS functions.
- *AMIGOS Installation Instructions and Release Notes* - contains all the information you need to get AMIGOS up and running on your computer.
- *GRAPH Reference Manual* - describes how to use the GRAPH software with AMIGOS to let your application make, store, retrieve and modify charts.

## 1.3 REFERENCE BOOKS

During AMIGOS's development, the books listed below have proven to be excellent resources for information about graphics.

- *Computer Graphics*. Written by Donald Hearn and M. Pauline Baker. Published in 1986 by Prentice-Hall, Inc.

- *Principles of Interactive Computer Graphics*, second edition. Written by William M. Newman, Robert F Sproull. Published in 1979 by McGraw-Hill Book Company.

- *Computer Graphics A Programming Approach*. Written by Steven Harrington. Published in 1983 by McGraw-Hill Book Company.

- *Fundamentals of Interactive Computer Graphics*. Written by J. D. Foley and A. Van Dam. Published in 1982 by Addison-Wesley Publishing Company, Inc.

- *Raster Graphics Handbook*. Written and published by Conrac Corporation in 1980.

- *PostScript Language Tutorial and Cookbook.* Written by Adobe Systems Incorporated. Published in 1985 by Addison-Wesley Publishing Company, Inc.

- *PostScript Language Reference Manual.* Written by Adobe Systems Incorporated. Published in 1985 by Addison-Wesley Publishing Company, Inc.

- *PostScript Language Program Design.* Written by Adobe Systems Incorporated. Published in 1988 by Addison-Wesley Publishing Company, Inc.

Alpha Microsystems' documents you may need to refer to are:

- *AMOS Monitor Calls Reference Manual*

- *AlphaBASIC User's Manual* or *AlphaBASIC PLUS User's Manual*

- *AlphaBASIC XCALL Subroutine User's Manual*

- *AlphaC User's Manual*

- *AlphaC AMOS Monitor Interface Manual*

- *AlphaC Release Notes*

- *AMOS Terminal System Programmer's Reference Manual*

## 1.4 HOW THIS BOOK IS ORGANIZED

The *AMIGOS Reference Manual* is organized into seven chapters and four appendices.

Chapter 2 - "General Concepts" introduces you to terms and ideas particular to graphics systems.

Chapter 3 - "AMIGOS Functions" describes status return codes, AlphaBASIC subroutine calls, control functions, graphical output and input, inquiry functions and error handling.

Chapter 4 "The Graphics Control Block" shows the format of the graphics control block and describes each of its fields.

Chapter 5 "The Graphics Device Driver" introduces you to the graphics device driver that translates AMIGOS's commands for the particular output device.

Chapter 6 "Reference Lists" contains a group of lists showing function name and syntax for specific reference needs. You'll find an alphabetic list by function name showing Assembler, AlphaBASIC, and AlphaC calling format, and lists organized by major function group.

Chapter 7 "Reference Sheets" is organized alphabetically by function name and provides you with the function's purpose and complete information on the calling sequence to use with Assembler, AlphaBASIC and AlphaC languages.

Appendix A - "Status Codes and Messages" lists the status codes and corresponding messages AMIGOS returns after call completion.

Appendix B - Defined Fill Area and Hatch Patterns" illustrates the fill area and hatch patterns already defined in AMIGOS.

Appendix C - "BMP Bitmap Image File Format" describes the BMP file's AMOS file header, bitmap image definition block, color palette definition block and image packing algorithm.

Appendix D - "Sample Programs" provides you with working programs in Assembler, AlphaBASIC, and AlphaC to illustrate AMIGOS's capabilities.

At the end of the appendices, a glossary contains many of the terms and definitions specific to graphics software generally and AMIGOS in particular.

## 1.5 PRINTING CONVENTIONS

Like other Alpha Micro documents, this book uses standard symbols and abbreviations to make the information easier to read and understand.

| SYMBOL | DESCRIPTION |
|---|---|
| `type` | This type face is used when illustrating the function format. For example:<br><br>`GOPWK gcb,status.` |
| {} | Optional elements in a function are enclosed within braces. When these symbols appear in a sample, they designate elements you may omit from the command line. |

# CHAPTER 2

# GENERAL CONCEPTS

This chapter discusses these general concepts: coordinate spaces, transformation, clipping, normalization and device coordinates.

## 2.1 COORDINATE SPACES

You may find it helpful in understanding the operation of AMIGOS to visualize two separate coordinate spaces. The first of these spaces is referred to as ***the world***. The world resides in the first quadrant of a Cartesian plane and has a range of 0 to 32767 in both X and Y directions. The user program performs all drawing with respect to the world coordinate space.

For example, when AMIGOS is directed to draw a line, the X and Y coordinates defined in the call serve as the endpoints for the line to be drawn. These coordinates must reside in the world space and, therefore, must be in the range of 0 to 32767. In Figure 2-1, a polyline is drawn in the world space using the coordinates 1000,2000 and 10000,15000 as endpoints.

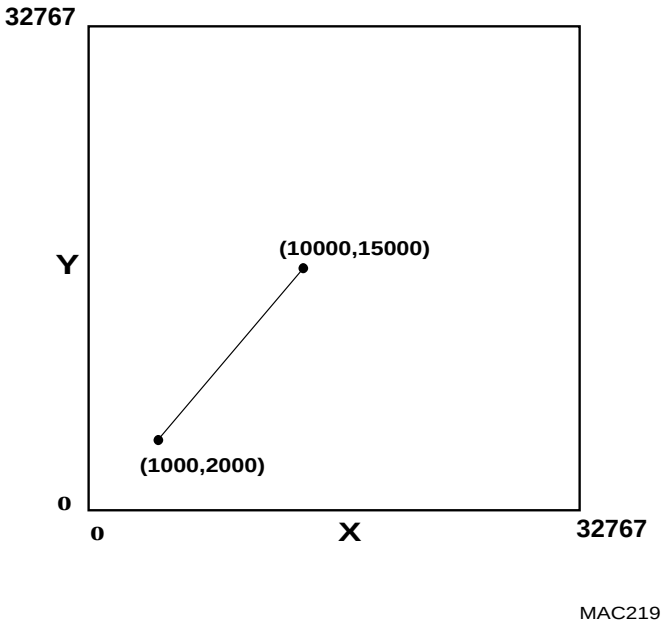



**Figure 2-1: The "World"**

Since the world coordinate space does not directly relate to an output device without some form of translation, its coordinates are often referred to as ***Virtual Device Coordinates*** (VDC). The user program draws in this virtual world without regard to the size or resolution of the output device.

The second coordinate space in AMIGOS is the ***Normalized Device Coordinate*** (NDC) space. This space also resides in the first quadrant of a Cartesian plane and has a range of 0 to 32767 in the X and Y directions. The NDC space is used by AMIGOS to translate the world drawn by the user into an image to be displayed on the output device. The user does not draw in the NDC space but it is presented here to provide an understanding of the transformations which occur between the user program and the output device.

AMIGOS defaults to displaying the entire world space in the entire NDC space. In this case, no transformation occurs, and the world space is the same as the NDC space. It is the job of the graphics device driver (GDV) to translate the NDC coordinates into the actual coordinates required for a specific output device. On output devices which do not have a square display area (most CRT devices) the entire NDC space may not be visible in the default mode (see Figure 2-2).

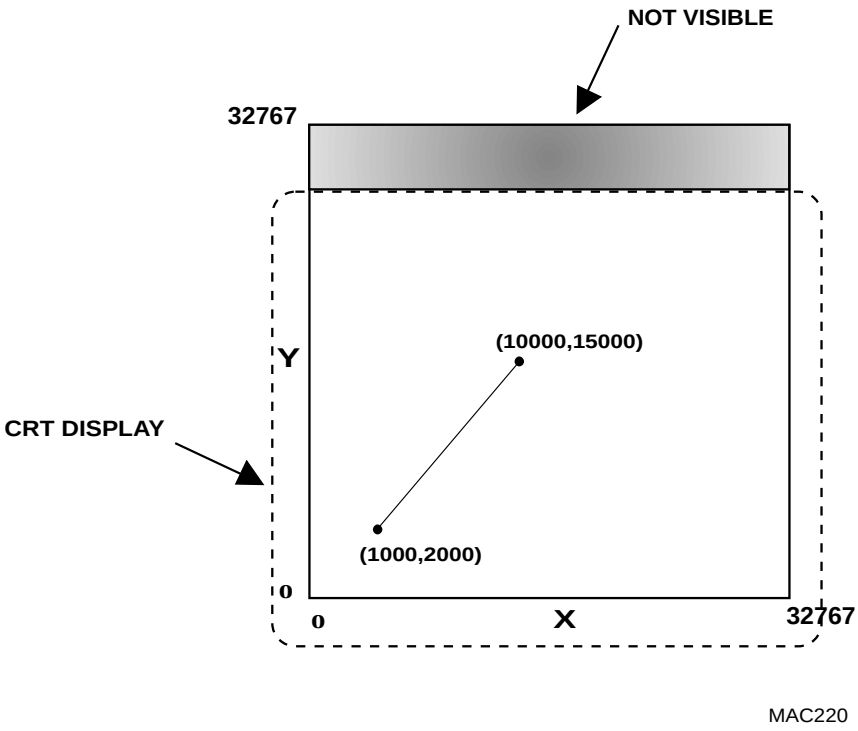



**Figure 2-2: Normalized Device Coordinate (NDC) Space**

## 2.2 WINDOWS

It may not always be desirable to display the entire world. The user may wish to view only a portion of the entire drawing. AMIGOS allows you to define a rectangular portion of the world space to be displayed. This partial world space is called a ***window*** (see Figure 2-3). When a window is defined in AMIGOS, that portion of the world space is mapped into NDC space for display. This normally results in all subsequent drawings being scaled or "zoomed up" for display.

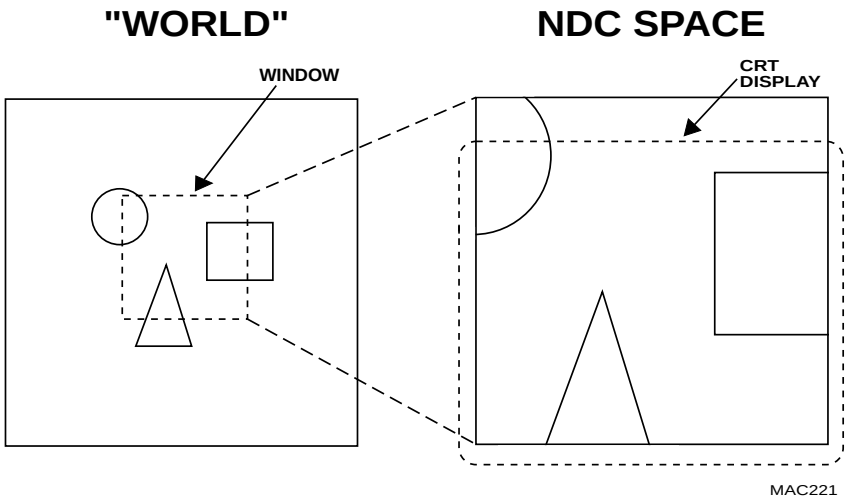



**Figure 2-3: A "Window"**

## 2.3 VIEWPORTS

AMIGOS defaults to scaling the window into the entire NDC space as shown in Figure 2-3. This may not always be desirable. It might be necessary to display the window in a smaller portion of the NDC space. We may, for example, wish to draw only in the upper right corner of the NDC space. AMIGOS allows you to define a rectangular portion of the NDC space in which to display the current window. This partial NDC space is called a ***viewport***. When a viewport is defined in AMIGOS, the currently defined window is mapped into the viewport area of the NDC space. This normally results in all subsequent drawings being scaled or "zoomed down" for display. Figure 2-4 shows the effect of defining a viewport. In this case, no window has been defined, so the window is equivalent to the entire world space. This function is useful in applications such as displaying multiple charts on a CRT screen. In this application, a viewport is defined followed by the display of the first chart. Then a second viewport is defined followed by the drawing of a second chart, and so on.

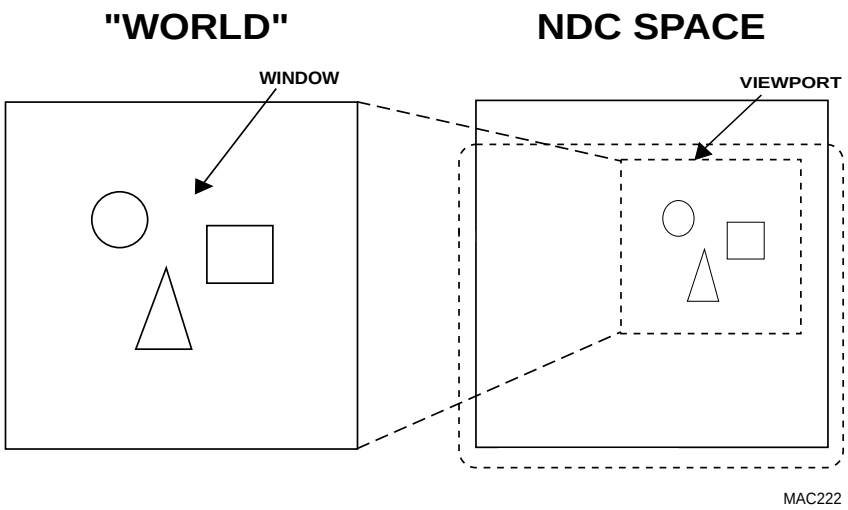



**Figure 2-4: Viewport**

## 2.4 TRANSFORMATION

The resulting scaling and movement of a drawing from the window in the world space to a viewport in the NDC space is called ***transformation***. Figure 2-5 details the effect of specifying a window and viewport and the resulting transformation of the drawing.

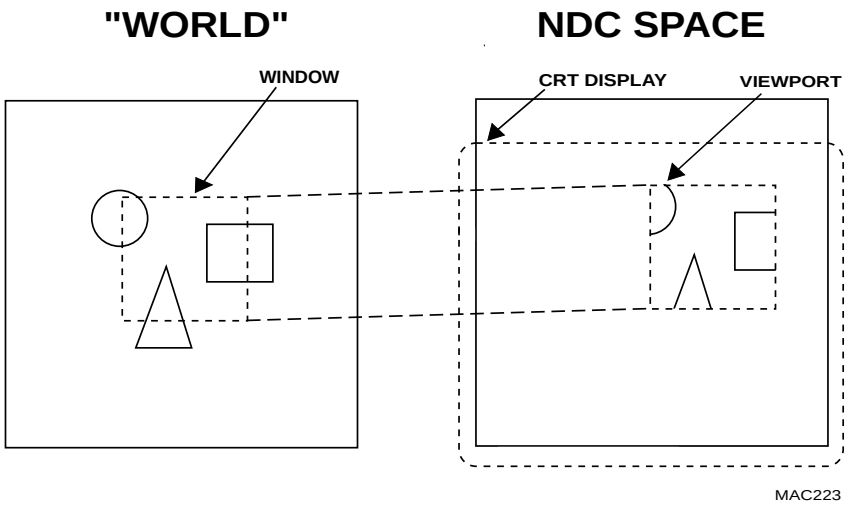



**Figure 2-5: Window and Viewport Transformation**

AMIGOS performs transformation on all output and input functions as the functions are executed. Objects which are already drawn on the output device are not transformed by AMIGOS when a new window or viewport are specified. If you want to transform drawings which are already displayed, it is your program's responsibility to redraw the objects with the new window and viewport in effect. You are free to specify any window and viewport desired, within the 0 to 32767 coordinate limits.

AMIGOS does not preserve the aspect ratio of the workstation while performing these transformations. It is possible, therefore, to produce a distorted image. When a workstation is opened, AMIGOS presets the window and viewport parameters to be equal to the entire world coordinate space.

## 2.5 CLIPPING

When a window is specified which is smaller than the world, all portions of the image falling outside the window are discarded. This process is referred to as ***clipping***. By performing clipping, AMIGOS ensures that no parts of the image are drawn outside the window boundary and no attempt is made to draw an image outside of the workstation display area.

## 2.6 NORMALIZATION AND DEVICE COORDINATES

The process of transformation and clipping results in a new image which is specified in Normalized Device Coordinates (NDC). All coordinates passed to a GDV during output are normalized. The GDV is responsible for translating these coordinates into Device Coordinates (DC) for the specific output device being used. This normally requires a translation from the 32K by 32K normalized coordinate space into the actual resolution of the device.

All coordinates returned to AMIGOS through the GDV by an input function are specified in NDCs. AMIGOS performs a reverse transformation which yields world coordinates to the user program.

# CHAPTER 3

# AMIGOS FUNCTIONS

This chapter describes AMIGOS functions. Topics included are: how to interface to AMIGOS via assembler, AlphaBASIC, and AlphaC subroutine calls; control functions; graphical output and input; inquiry functions and error handling.

## 3.1 INTERFACING WITH AMIGOS

Depending on the language used in your application program, you will communicate with AMIGOS in one of three different language bindings. Each binding calls the same routines in AMIGOS, but each binding has a unique method of passing arguments to and from the routines. In general, AMIGOS is called specifying a particular function to be performed. Along with the function, your program supplies any necessary arguments required by the function. Upon return, AMIGOS provides a status indicating whether the call was successful.

### 3.1.1 Assembler Binding

The assembly language binding provides a closely coupled, efficient interface to AMIGOS. Your program maintains call-specific argument blocks and variables for use by AMIGOS. The calls themselves are macro expansions which cause a subroutine call to the AMIGOS dispatch vector in the System Communication Area. In most calls, your program may specify a return status argument as part of the call. The assembler binding is available by SEARCHing for AMGSYM.UNV at the head of the program. This module defines all of the calls and associated data structure items.

### 3.1.2 AlphaC Binding

The AlphaC language interface to AMIGOS is very similar in structure to the assembly language binding. Your program includes AMIGOS.H in the program to define the calls and data structures required by AMIGOS. When the compiled object code is linked with the AMGCLB.LIB library, the resulting program contains the interface to the AMIGOS routines. The data structures required by individual calls are detailed in the reference sheets section of this manual in Chapter 7. All calls return status as the returned variable. This allows your program to store the status or use the conditional call method as shown in the AlphaC example in Appendix D.

### 3.1.3∞AlphaBASIC Binding

You may use the ++INCLUDE statement to include the file AMGSYM.BSI in an AlphaBASIC program to use AMIGOS from an AlphaBASIC subroutine level. The subroutine, AMGSBR.SBR (for AlphaBASIC) or AMGSBR.XBR (for AlphaBASIC PLUS), includes all calls to the AMIGOS routines. The interface to AMIGOS from AlphaBASIC is somewhat different in concept from the Assembler and AlphaC bindings. A major difference exists in the area of memory allocation. AMIGOS requires a Graphics Control Block (GCB), explained in Chapter 4, to perform operations on a workstation. In addition to the GCB, AMIGOS requires a block of "work" memory in which to perform graphics functions. This block, called the ***dynamic impure area***, is variable in size, dependent upon the type of workstation being used. Since AlphaBASIC is not able to allocate memory dynamically, it is necessary for the program to contain a MAP statement which defines an unformatted variable large enough to include both the GCB and the dynamic impure area. The following statement allocates a 32,000 byte area for use by AMIGOS.

```
MAP1 GCB,X,32000
```

A GCB, such as that described above is necessary for each open workstation. Some raster type workstations may require more than 32K of dynamic impure space. If this is the case, the MAP statements must be broken into multiple statements such as:

```
MAP1 GCB
   MAP2    GCBA,X,32000
   MAP2    GCBB,X,30000
```

The above statements allocate a 62,000 byte area for use by AMIGOS.

Certain calls require additional arguments passed as a binary point array. The following format defines an X-Y point array:

```
MAP1 POINT'ARRAY
   MAP2 POINT'COUNT,B,2          !total count of points
   MAP2 POINTS(n)                !n = maximum number of points
      MAP3 X'POINT,B,2           ! X coordinate
      MAP3 Y'POINT,B,2           ! Y coordinate
```

### 3.2∞CONTROL FUNCTIONS

AMIGOS contains several control functions used to begin, terminate, and control the use of a graphics workstation from the applications software. Each workstation to be accessed through AMIGOS must first be opened by the Open Workstation (GOPWK) function. When you are through using the workstation you terminate its use with the Close Workstation (GCLWK) function.

Additional control functions allow clearing (initialization) of the workstation display surface, updating of the workstation display for synchronization purposes, and performance of special workstation dependent functions. Whenever a workstation has

been opened for use by a program, it is represented within the program by a data structure known as the ***Graphics Control Block*** (GCB). This control block, which is allocated by the application software and therefore resides in the user's memory partition, is similar in concept to the Dataset Driver Block (DDB) used by the AMOS file system to refer to data files. The Graphics Control Block is discussed further in Chapter 4.

## 3.3 GRAPHICAL OUTPUT

The graphical information AMIGOS generates and routes to the workstation is built up of basic pieces called ***output primitives***. AMIGOS provides six output primitives:

- Polyline (GPL) - AMIGOS generates a set of connected lines defined by a point sequence.
- Polymarker (GPM) - AMIGOS generates symbols of one type centered at given positions.
- Text (GTX) - AMIGOS generates a character string at a given position.
- Fill Area (GFA) - AMIGOS generates a polygonal area which may be hollow, filled with a solid color, or filled with a pattern.
- Bitmap (GBM) - AMIGOS generates an array of pixels with individual colors.
- Generalized Drawing Primitive (GDP) - AMIGOS addresses special geometrical output capabilities of workstations, such as drawing spline curves, circular arcs, elliptic arcs, etc. The objects are characterized by a primitive type, a set of points, plus other data.

Each output primitive potentially has two types of attributes: geometric and non-geometric. These attribute types determine the exact appearance of the output primitive.

The values of these attributes are set and stored by AMIGOS in the workstation state list. A separate AMIGOS function is provided for each primitive attribute to allow the application program to specify the value of an attribute without unnecessarily specifying the values of other attributes. During the creation of an output primitive (that is, when one of the AMIGOS output primitive functions is invoked) these values are bound to the primitive and cannot be changed afterward.

Geometric attributes control the geometric aspects of primitives; these are aspects affecting the shape or size of a primitive (for example, Character Height (GSCHH) for Text (GTX). Each geometric attribute is defined separately for each primitive and a primitive may have no, one, or many geometric attributes.

Non-geometric attributes control primitive aspects which do not affect the shape or size of a primitive but only affect its appearance (for example, Linetype (GSPLT) for Polyline (GPL), or Color Index (GSFAC) for all primitives except Bitmap (GBM)).

There is a separate attribute for each non-geometric aspect. As with the attributes controlling the geometric aspects, these attributes are workstation independent. Each of these attributes applies to only one primitive type.

As indicated above, Generalized Drawing Primitive (GDP) and Bitmap (GBM) do not have corresponding attributes. The GDP uses the most appropriate of the individual attributes for each GDP function. GBM contains color index information as part of its definition but has no other non-geometric aspects.

The following table shows the attributes which apply to each output primitive.

| PRIMITIVE TYPE | ATTRIBUTE |
|---|---|
| Polyline | Linetype<br>Linewidth<br>Polyline Color Index |
| Polymarker | Marker Type<br>Marker Size<br>Polymarker Color Index |
| Text | Text Font<br>Text Color Index<br>Character Height<br>Character Rotation |
| Fill Area | Fill Area Interior Style<br>Fill Area Style Index<br>Fill Area Color Index |
| Bitmap | none |
| Generalized Drawing Primitive | Polyline or Fill Area Attributes |

### 3.3.1 Polyline Attributes

Polyline has no geometric attributes. The representation of polyline at the workstation is controlled by these individually specified attributes:

- Linetype (GSPLT)
- Linewidth (GSPLS)

- Polyline Color Index (GSPLC)

The Linetype (GSPLT) specifies a sequence of line segments and gaps which are repeated to draw a polyline. Whether this sequence is restarted or continued at the start of the polyline or at each vertex of a polyline is workstation dependent.

Linetype 1 is solid, Linetype 2 is dashed, Linetype 3 is dotted, and Linetype 4 is dashed-dotted. Every workstation supports these four linetypes in a recognizable fashion. Linetypes greater than 4 may be available but their styles are workstation dependent.

The Linewidth (GSPLS) is specified as a nominal linewidth in world coordinates. This value is mapped by the workstation to the nearest available linewidth. For solid linetypes, a line width specification larger than the workstation can support results in AMIGOS drawing the wide solid line as a filled area. Multiple segments in a polyline are mitred during the wide line emulation.

The Color Index (GSPLC) specifies which entry in the workstation's color table will be used to display the polyline.

### 3.3.2 Polymarker Attributes

Polymarker (GPM) has no geometric attributes. The representation of polymarker at the workstation is controlled by the individual polymarker attributes:

- Marker Type (GSPMT)
- Marker Size (GSPMS)
- Polymarker Color Index (GSPMC)

Markertypes 1 to 5 are defined as:

Markertype 1 = dot
Markertype 2 = plus sign
Markertype 3 = asterisk
Markertype 4 = circle
Markertype 5 = diagonal cross

Each markertype is centered on the position it is identifying. Every workstation supports these five markertypes in a recognizable fashion. Markertypes greater than 5 may be available but their styles are workstation dependent.

The Markersize is specified as a nominal size in world coordinates. This value is mapped by the workstation to the nearest available size. Markersize 1 is always displayed as the smallest displayable dot.

The Color Index specifies which entry in the workstation's color table will be used to display the polymarker.

### 3.3.3 Text Attributes

Text has these geometric attributes:

- Character Height (GSCHH)
- Character Rotation (GSCHR)

The representation of text at the workstation is controlled by the individual attributes Text Font (GSTXF) and Text Color Index (GSTXC).

Appearance of text on the workstation is controlled by the aspects Character Height (GSCHH) and Character Rotation (GSCHR). However, the use of these values in displaying text is determined by the setting of the text font. GSCHH specifies the nominal height of a capital letter character. The GSCHR specifies the angle at which the text will be displayed relative to a provided origin point. This angle is specified as a counterclockwise rotation in tenths of degrees.

The text font value is used to select a particular font on the workstation. Every workstation supports at least one font that is capable of generating a graphical representation of the characters defined in ANSI X3.4-1977, commonly known as ASCII. This is font number 1.

AMIGOS also provides a set of internally generated stroke fonts numbered 1001 to 1009. Font 1001, a Simplex Roman stroke font is always available. Fonts 1002 through 1009 are defined as follows:

| FONT # | TYPE | FILE |
|---|---|---|
| 1002 | Simplex Script | SIMSCR.FNT |
| 1003 | Complex Roman | COMROM.FNT |
| 1004 | Complex Italic | COMITL.FNT |
| 1005 | Complex Script | COMSCR.FNT |
| 1006 | Duplex Roman | DUPROM.FNT |
| 1007 | Triplex Roman | TRIROM.FNT |
| 1008 | Triplex Italic | TRIITL.FNT |
| 1009 | Gothic Roman | GOTHIC.FNT |

These fonts must reside in user or system memory to be used. If a font above 1001 is specified and not found in memory, AMIGOS defaults to font 1001 for subsequent text operations. If a font below 1001 is specified and the workstation does not support that font, the default workstation font (font 1) is used.

### 3.3.4 Fill Area Attributes

The representation of fill area at the workstation is controlled by the individual attributes:

- Fill Area Interior Style (GSFAS)
- Fill Area Style Index (GSFAI)
- Fill Area Color Index (GSFAC)

The fill area interior style is used to determine in what style the area should be filled. It has the following values:

- Hollow - No filling, but draw the bounding polyline, using the Fill Area Color Index currently selected. The Linetype and Linewidth are workstation dependent, but will normally default to the current polyline type and width.
- Solid - Fill the interior of the polygon using the fill area color index currently selected.
- Pattern - Fill the interior of the polygon using the fill area style index currently selected as an index into the pattern table. In this context, the fill area style index is sometimes referred to as the pattern index.
- Hatch - Fill the interior of the polygon using the fill area style index currently selected as an index into the internal hatch table. The area will be filled with a hatch pattern generated with polylines, in order to simulate a fill pattern. In this context, the fill area style index is sometimes referred to as the hatch index.

For interior style Pattern, the pattern index is selected from workstation dependent patterns. Certain of these patterns have been pre-defined and are shown in Appendix B, "Fill Area and Hatch Patterns." Other patterns may be available on specific workstations.

Interior style Hollow is available on all workstations. It is workstation dependent which of the interior styles Solid and Pattern are available.

### 3.3.5 Bitmap Attributes

Bitmap (GBM) has no attributes associated with it. However, an array of color indices, which are pointers into the color table, is part of the definition of a bitmap.

### 3.3.6 Generalized Drawing Primitive Attributes

Generalized Drawing Primitive (GDP) has no explicit geometric attributes. Such information may be specified in the GDP function. The representation of the GDP on the workstation is controlled by Polyline or Fill Area attributes. The sets of attributes most appropriate for a given GDP function are selected by the GDP function.

### 3.3.7 Color

Color is specified in a number of different situations. It may be an aspect of a primitive such as Polyline. It may be part of a pattern for Fill Area Interior Style (GSFAS), in which case an array of colors is specified, or it may part of a primitive itself, namely Bitmap (GBM), when an array of colors is also specified. In each case the color is specified as an index into a color table on the workstation. On each workstation, there is one color table into which all of the color indices point.

The size of the color table is workstation dependent but entries 0 and 1 always exist. Entry 0 corresponds to the background color. The background color is the color of the display surface after it has been cleared. Entry 1 is the default foreground color and entries higher than 1 correspond to alternative foreground colors. The specified color is mapped to the nearest color available on the workstation. On some workstations it may not be possible to change the background color (such as being unable to change the color of the paper on a pen plotter) and in this case the mapping of a specific color to the nearest available for background color may be different from the mapping of the same color for the foreground colors.

The color representation is set through the use of the HLS (Hue, Lightness, saturation) system, as defined in the *AMOS Terminal System Programmer's Manual*, or RGB (Red, Green, Blue) depending on the current color mode. Briefly, all colors may be represented as a combination of hue, lightness, and saturation. Hue is specified as an angle of rotation about the vertical axis of the color cone model, and ranges from 0 to 360 degrees. Lightness and saturation are specified as a percentage of total with a range of 0 to 100 percent. In RGB mode, the color is represented by a mixture of red, green and blue components, each specified in the range 0 to 255. A value of 0 indicates no color, while a value of 255 indicates full color.

Some workstations are not capable of displaying colors (for example, workstations only capable of displaying colors with equal red, green, and blue intensities or workstations capable of displaying colors which are different intensities of the same color); these are called monochrome workstations. Whether a workstation is capable of color is recorded in the "color available" bit in the workstation descriptor field. On monochrome workstations, the intensity is computed from the color respresentation desired. This is normally a gray scale value corresponding to the NTSC color standard, consisting of 30% red, 59% green and 11% blue.

The workstation will select a color from a palette which most closely matches the specified color. On a monochrome display, the workstation driver (GDV) determines the closest representation of the desired color.

## 3.4 GRAPHICAL INPUT

AMIGOS provides the capability of graphical input in two classes:

- Locator (GRQLC) - provides a position in world coordinates.
- Valuator (GRQVL) - provides an integer value.

The physical function of each of these input classes is workstation dependent. For example, a Request Locator (GRQLC) function performed on a Tektronix 4105 terminal might use the joydisk as the input device, while the same function performed on an IBM PC might use a mouse or cursor keys. The implementation of the function is controlled by the GDV for a specific workstation.

## 3.5 INQUIRY FUNCTIONS

Inquiry functions return information about the current state of the graphics workstation, including device capabilities and current attributes. This class of functions is also used in error reporting.

## 3.6 ERROR HANDLING

AMIGOS returns the status of each call in a language binding-dependent manner. From assembler level, the Z-flag is reset if any error occurs. You can specify a return status register on each call, or the error code can be extracted directly from the GCB. AMIGOS also provides the Inquire Error (GQERR) function to return an error message corresponding to the error code in GC.ERR in the GCB.

The action AMIGOS performs when an error occurs depends on the specific function being performed and on the setting of the return on error (GC$ERC) flag in the GCB. If the flag is not set, AMIGOS will cause the user program to EXIT to AMOS command level. If the flag is set, AMIGOS will perform the requested function in as normal a manner as possible to the point that the error occurs. In this mode, it is the user's responsibility to detect all errors and provide suitable reporting and recovery.

AMIGOS will display an error message on the user's terminal unless the bypass error print (GC$BYP) flag has been set in the flags word.

Error codes resulting from AMIGOS calls start at 512 decimal and extend upward. Error codes in the range 0 to 255 decimal are standard AMOS file system errors which are duplicated in GC.ERR for convenience. This approach to error encoding allows the programmer to specify an alternate output DDB in GC.OUT and condition the DDB to return on error conditions. The programmer may determine whether an error returned by AMIGOS is related to file handling or graphics by testing the value returned.

## 3.7 STATUS RETURN CODES

Each graphics function returns a 16-bit status code upon its completion. This 16-bit status code is used to notify the application software of any errors or warnings that may occur during the use of the graphics workstation. (See Appendix A, "Status Codes and Messages," for more information.)

If the graphics operation is successful, this status code will contain a zero. For assembly language users, the Z-flag status bit will also be set. All error conditions will return a non-zero error code.

# CHAPTER 4

# THE GRAPHICS CONTROL BLOCK

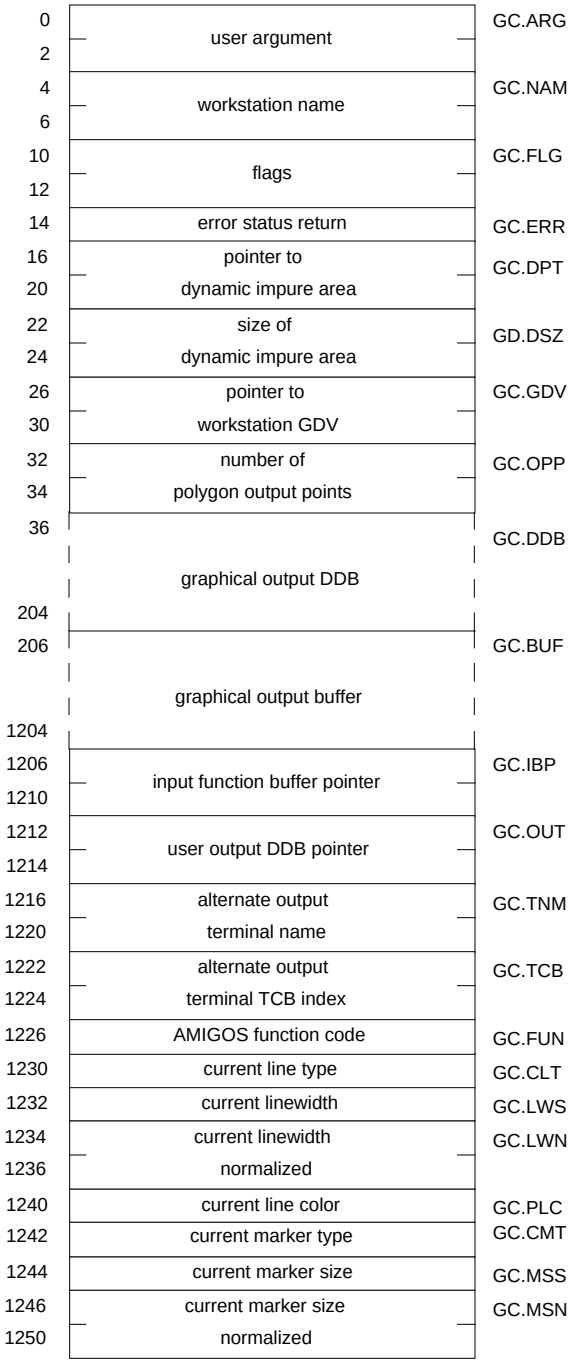
Whenever a workstation has been opened for use by a program, it is represented within the program by a data structure known as the ***Graphics Control Block*** (GCB). This control block, which is allocated by the application software and therefore resides in the user's memory partition, is similar in concept to the Dataset Driver Block (DDB) used by the AMOS file system to refer to data files.

Before you can perform any graphics function, you must first allocate a GCB within your partition. This data structure has a size (defined in AMGSYM.UNV) of GC.SIZ bytes. AMIGOS requires a larger work space than this initial GCB in order to perform graphics input and output. The graphics device driver (GDV) may also require a work area for temporary storage of variables during operation. For this reason, the GCB contains a pointer to an additional memory area which must be allocated in the user partition. This area is called the ***dynamic impure area***.

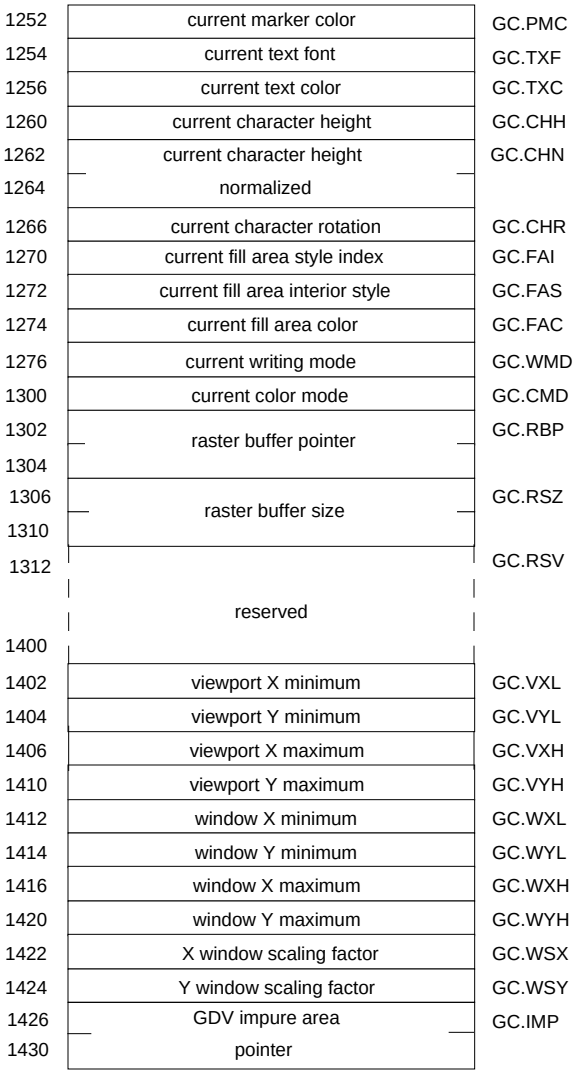
If the user program allows standard AMOS memory allocation techniques (GETMEM), it may simply perform the AMIGOS GOPWK (open workstation) call, with a value of zero in the GCB dynamic impure area pointer (GC.DPT). This will cause AMIGOS to allocate the dynamic impure area on its own. If the user program allocates its own memory modules, the dynamic impure area pointer (GC.DPT) is initialized to point to the user's work area before the Open Workstation call.

Your program may determine the required size of the dynamic impure area prior to the open workstation process through use of the Inquire Dynamic Impure Size (GQDSZ) call. The internal format of the graphics control block is illustrated in the following table.

## Graphics Control Block - Internal Format




MAC227




MAC229

### 4.1 USER ARGUMENT (GC.ARG)

GC.ARG is a 32-bit field used to pass arguments such as values, pointers, etc., to AMIGOS routines. This field is normally updated by the graphical calls and therefore need not be accessed directly by the programmer.

### 4.2 SYMBOLIC WORKSTATION NAME (GC.NAM)

GC.NAM is a 32-bit field containing the symbolic name of the graphical workstation that this GCB is representing. The symbolic name is from one to six alphanumeric characters, packed RAD50. If this field contains a binary zero, the currently attached terminal is assumed to be the intended graphical workstation. If the terminal is not capable of performing as a graphical workstation, an error will be returned.

### 4.3 FLAGS (GC.FLG)

GC.FLG is a 32-bit field containing various status flags used by the Graphics Device Driver. The flags are:

GC$ERC Return on error condition

GC$BYP Bypass printing of error messages

GC$FLA This flag is used by Graphics Device Drivers (GDV) to determine whether the current function is part of a filled area. This flag should not be modified.

GC$RSE This flag indicates that an error has occurred during raster device processing and no further further operations should take place. This flag is used internally in AMIGOS and should not be modified.

### 4.4 ERROR RETURN (GC.ERR)

GC.ERR is a 16-bit field updated at the completion of each graphical operation to contain the status code indicating the success or failure of the operation. This field will contain zero when the operation was successful. If the operation failed, a status code (defined in Appendix A) will be returned.

### 4.5 POINTER TO DYNAMIC IMPURE AREA (GC.DPT)

GC.DPT is a 32-bit field pointing to the dynamic impure area required by AMIGOS for intermediate storage. It is either allocated by the user program or by AMIGOS during the GOPWK Open Workstation call.

### 4.6 SIZE OF DYNAMIC IMPURE AREA (GC.DSZ)

GC.DSZ is a 32-bit field containing the size of the dynamic impure area required by AMIGOS for intermediate storage. It is set by the GQDSZ (Inquire Dynamic Impure Size) function or the GOPWK (Open Workstation) function.

### 4.7 POINTER TO WORKSTATION GDV (GC.GDV)

GC.GDV is a 32-bit field pointing to the Graphical Device Driver used by the specified workstation.

### 4.8 NUMBER OF POLYGON OUTPUT POINTS (GC.OPP)

GC.OPP is a 32-bit field containing the maximum number of polygon output points which may be generated in a single fill area (GFA) command. This value may be set by the user program prior to the GQDSZ Inquire Dynamic Impure Size call to allocate a larger polygon area in the dynamic impure zone.

### 4.9 I/O DDB (GC.DDB)

GC.DDB is used by the GDV to perform I/O on the graphical workstation.

### 4.10 I/O BUFFER (GC.BUF)

GC.BUF is a 512-byte field used in conjunction with the I/O DDB (GC.DDB) when performing I/O on the graphical workstation.

### 4.11 INPUT FUNCTION BUFFER POINTER (GC.IBP)

GC.IBP is a 32-bit field used as a pointer to a buffer used for input functions. This field also receives a single character input in the input calls.

### 4.12 USER OUTPUT DDB INDEX (GC.OUT)

GC.OUT is a 32-bit field used as a pointer to a user DDB which is used for all graphical output. This allows output redirection.

### 4.13 ALTERNATE OUTPUT TERMINAL NAME (GC.TNM)

GC.TNM is a 32-bit field used to define an alternate output terminal during the Open Workstation (GOPWK) function. The six character terminal name is packed RAD50 in this field.

### 4.14 ALTERNATE TERMINAL OUTPUT TCB INDEX (GC.TCB)

GC.TCB is a 32-bit field used as a pointer to the terminal control block corresponding to the alternate terminal name in TC.TNM during the Open Workstation call. The graphics device driver (GDV) uses this index to redirect output to an alternate terminal.

### 4.15 CURRENT FUNCTION CODE (GC.FUN)

GC.FUN is a 16-bit field defining the current AMIGOS function and is used to internally keep track of the current call level.

### 4.16 CURRENT LINE TYPE (GC.CLT)

GC.CLT is a 16-bit field defining the currently active polyline type, as shown in the following table.

| TYPE | MEANING |
|---|---|
| 1 | solid line |
| 2 | dashed line |
| 3 | dotted line |
| 4 | dashed-dotted line |
| >4 | device dependent |

### 4.17 CURRENT LINEWIDTH (GC.LWS)

GC.LWS is a 16-bit field defining the currently active linewidth.

### 4.18 CURRENT LINEWIDTH NORMALIZED (GC.LWN)

GC.LWN is a 32-bit field defining the currently active linewidth in normalized device coordinates (NDC).

### 4.19 CURRENT POLYLINE COLOR INDEX (GC.PLC)

GC.PLC is a 16-bit field defining the currently active color index associated with polyline commands.

### 4.20 CURRENT MARKER TYPE (GC.CMT)

GC.CMT is a 16-bit field defining the currently active polymarker type. The valid marker types are:

| TYPE | MEANING |
|---|---|
| 1 | dot (.) |
| 2 | plus (+) |
| 3 | star (*) |
| 4 | circle (O) |
| 5 | cross (X) |
| >5 | device dependent |

### 4.21 CURRENT MARKER SIZE (GC.MSS)

GC.MSS is a 16-bit field defining the currently active polymarker size.

### 4.22 CURRENT MARKER SIZE NORMALIZED (GC.MSN)

GC.MSN is a 32-bit field defining the currently active polymarker size in normalized device coordinates (NDC).

### 4.23 CURRENT POLYMARKER COLOR INDEX (GC.PMC)

GC.PMC is a 16-bit field defining the currently active color index associated with polymarker commands.

### 4.24 CURRENT TEXT FONT (GC.TXF)

GC.TXF is a 16-bit field defining the currently active text font.

### 4.25 CURRENT TEXT COLOR INDEX (GC.TXC)

GC.TXT is a 16-bit field defining the currently active color index associated with text.

### 4.26 CURRENT CHARACTER HEIGHT (GC.CHH)

GC.CHH is a 16-bit field defining the currently active character height.

### 4.27 CURRENT CHARACTER HEIGHT NORMALIZED (GC.CHN)

GC.CHN is a 32-bit field defining the currently active character height in normalized device coordinates (NDC).

### 4.28 CURRENT CHARACTER ROTATION (GC.CHR)

GC.CHR is a 16-bit field defining the currently active character rotation in tenths of degrees.

### 4.29 CURRENT FILL AREA STYLE INDEX (GC.FAI)

GC.FAI is a 16-bit field defining the currently active fill area style index.

### 4.30 CURRENT FILL AREA INTERIOR STYLE (GC.FAS)

GC.FAS is a 16-bit field defining the currently active fill area interior style.

### 4.31 CURRENT FILL AREA COLOR INDEX (GC.FAC)

GC.FAC is a 16-bit field defining the currently active color index associated with fill area commands.

### 4.32 CURRENT WRITING MODE (GC.WMD)

GC.WMD is a 16-bit field defining the currently active writing mode.

The writing modes are:

| TYPE | MEANING |
|---|---|
| 0 | replace (G$WREP) |
| 1 | exclusive or (G$WXOR) |
| 2 | logical and (G$WAND) |
| 3 | logical or (G$WOR) |
| >3 | device dependent |

### 4.33 CURRENT COLOR MODE (GC.CMD)

GC.CMD is a 16-bit field defining the currently active color mode. The color modes can be set to either HLS (hue, lightness, saturation) or RGB (red, green, blue). A value of 0 indicates HLS mode, a value of 1 indicates RGB mode.

### 4.34 RASTER BUFFER POINTER (GC.RBP)

GC.RBP is a 32-bit field containing an index to a buffer maintained by AMIGOS for raster type devices. The user program may place a raster buffer index into this location prior to opening a workstation. This location must not be modified while a workstation is open.

### 4.35 RASTER BUFFER SIZE (GC.RSZ)

GC.RSZ is a 32-bit field containing the size of a buffer maintained by AMIGOS for raster type devices. The user program may place a raster buffer size into this location prior to opening a workstation. This location must not be modified while a workstation is open.

### 4.36 RESERVED (GC.RSV)

GC.RSV is reserved for future expansion.

### 4.37 VIEWPORT X MINIMUM (GC.VXL)

GC.VXL is a 16-bit field containing the minimum X coordinate of the currently active viewport.

### 4.38 VIEWPORT Y MINIMUM (GC.VYL)

GC.VYL is a 16-bit field containing the minimum Y coordinate of the currently active viewport.

### 4.39 VIEWPORT X MAXIMUM (GC.VXH)

GC.VXH is a 16-bit field containing the maximum X coordinate of the currently active viewport.

### 4.40 VIEWPORT Y MAXIMUM (GC.VYH)

GC.VYH is a 16-bit field containing the maximum Y coordinate of the currently active viewport.

### 4.41 WINDOW X MINIMUM (GC.WXL)

GC.WXL is a 16-bit field containing the minimum X coordinate of the currently active window.

### 4.42 WINDOW Y MINIMUM (GC.WYL)

GC.WYL is a 16-bit field containing the minimum Y coordinate of the currently active window.

### 4.43 WINDOW X MAXIMUM (GC.WXH)

GC.WXH is a 16-bit field containing the maximum X coordinate of the currently active window.

### 4.44 WINDOW Y MAXIMUM (GC.WYH)

GC.WYH is a 16-bit field containing the maximum Y coordinate of the currently active window.

### 4.45 WINDOW X SCALING FACTOR (GC.WSX)

GC.WSX is a 16-bit field containing the X component of the scaling factor used to transform output primitives to the current window and viewport.

### 4.46 WINDOW Y SCALING FACTOR (GC.WSY)

GC.WSY is a 16-bit field containing the Y component of the scaling factor used to transform output primitives to the current window and viewport.

### 4.47 IMPURE AREA POINTER FOR GDV (GC.IMP)

GC.IMP is a 32-bit field used as a pointer to an impure zone within the dynamic impure area which is used by the graphics device driver (GDV) to store information unique to the particular graphics workstation in use.

# CHAPTER 5

# THE GRAPHICS DEVICE DRIVER

AMIGOS provides a device independent interface to different graphics devices by insulating the programmer from any differences between devices. It does this by presenting a common, uniform appearance to the programmer, independent of the requirements of a specific device. It is the job of the Graphics Device Driver (GDV) to perform any necessary translation between the AMIGOS commands given by a program and the actual code sequences required by a particular graphics device. As we will see, different types of graphics devices require markedly different support.

AMIGOS presents a common interface which has vector oriented commands (such as the polyline function), raster oriented commands (such as the bitmap function), as well as commands which contain a bit of both (such as the fill area function). AMIGOS provides these different types of commands so that you can use the command best suited to your application.

Graphics display devices also come in both vector and raster command types. Some, such as pen plotters, are only capable of drawing vectors. Others, such as the AM-72 color graphics terminal, accept both vector commands and raster commands. Still others, such as dot-matrix printers, are only capable of accepting rasters.

To deal with the different types of devices efficiently, AMIGOS separates them into two distinct classes: ***vector*** and ***raster***. While you still use the same AMIGOS commands with both types, the way in which AMIGOS internally deals with these two types, and the internal function of the GDVs, differs significantly. While you can ignore these differences and still get useful graphics output, knowing the different ways that AMIGOS treats these devices can help you optimize the performance of your graphics display.

Remember that the type of commands, vector or raster, used by a given device have little to do with the actual method of display used. Just about all display devices you will run into, except for pen plotters, are actually raster devices internally. Such raster devices, however, may accept vector oriented commands; examples of this are the AM-72 or a PostScript printer. AMIGOS is sensitive to the command type, not the actual display method.

## 5.1 AN OVERVIEW OF THE GDV

As mentioned above, the primary role of the GDV is to translate the stream of AMIGOS commands issued by the calling program into whatever set of commands are required by the device the GDV is supporting. It does this by providing two different types of resources to AMIGOS.

The first of these is the GDV header. This header contains many different items which describe the characteristics of the device (how many colors it supports, its resolution, etc.) and the capabilities of the GDV itself (whether it supports area fills, whether it supports text, etc.). This information is used both by the calling program and by AMIGOS. While the calling program may use such information to decide whether or not to use color or some other such feature, AMIGOS also uses this information to determine how best to emulate those display features that the GDV does not directly support.

The second resource within the GDV is the GDV routines themselves. For each AMIGOS call, there is a corresponding routine within the GDV. Each time the AMIGOS function is called, the corresponding routine within the GDV is called. This allows the GDV to perform any special action it needs to for each AMIGOS call.

AMIGOS is capable of emulating most of the graphics functions itself, given only the basic capability of drawing a line. If a GDV has no more capability than this, AMIGOS will be able to draw text, fill areas, draw hatch patterns, and otherwise perform most common graphics operations. Devices that have direct support for these functions, however, may be able to improve display performance by not relying on AMIGOS to do emulation.

AMIGOS relies heavily on the information in the GDV header. While it may not seem important to exactly calculate a parameter such as the size of a pixel in micrometers, this value, along with all the others, is an important part of the information used by AMIGOS.

Sample source code to several different types of GDVs is provided with AMIGOS. Many times new devices can easily be supported by simply modifying existing GDVs, rather than trying to start from scratch.

### 5.1.1 GDV Memory Usage

Whenever you open a workstation, you must allocate memory space for a graphics control block (GCB). In addition to the memory needed for this GCB, each individual GDV may require additional impure memory space. You can determine the amount of impure memory space a specific GDV needs by using the GQDSZ call.

Raster GDVs also need additional buffer memory, as discussed below.

Making sure you have sufficient memory allocated is particularly important in environments such as AlphaBASIC, which do not support dynamic memory allocation.

### 5.1.2 GDV Disk Usage

A GDV need not use any disk space at all. However, most printer and plotter type GDVs perform their output to a disk file, rather than directly to the device, allowing for spooling of output in a multi-user environment. You must make sure that there is enough disk space available for these output files on your login disk when using such GDVs.

Raster GDVs need additional disk space for buffering, as discussed below.

## 5.2 VECTOR GDVS

Vector GDVs are used for those devices which accept line drawing commands. Because the devices that support vector style commands are often fairly intelligent devices, most of the work in a vector GDV consists of scaling coordinates to device coordinate space and outputting specific command sequences to the vector device.

## 5.3 RASTER GDVS

Raster GDVs are used for those devices which have no built-in line drawing commands, but instead are only capable of displaying a raster image. Examples of devices that use raster GDVs are dot-matrix printers and non-PostScript laser printers.

To support such devices, AMIGOS builds a copy of the image in memory, performing all line drawing, area filling, and other commands itself, in conjunction with the GDV. Only when the display operation is completed is this raster image sent to the display device.

Raster GDVs require even less internal support than vector GDVs. A typical raster GDV will consist primarily of calls to AMIGOS library functions which perform all of the work related to laying down the pixels in the raster buffer. The primary job of the raster GDV is, upon the Close Workstation call, to convert the contents of the AMIGOS raster buffer to the device specific code required to render the raster image on the display device.

Because so much of the raster image generation function performed by AMIGOS is based on device specific parameters, it is (if possible) even more critical that the GDV header values be properly set than it is with vector GDVs. Improper setting of these values will most often lead to completely unpredictable display results.

### 5.3.1 Raster GDV Memory Requirements

In order to have space in which to build this raster image, AMIGOS uses a combination of memory and disk buffers. Because this raster image can be quite large (typically well over one megabyte), you must be sure that you have adequate memory and disk space available before using a raster GDV.

AMIGOS by default uses an 8K buffer within your partition (in addition to the GCB and any impure space required by the particular GDV), paging the remainder of the raster buffer out to disk. This allows image generation in a small memory partition, but causes a great deal of disk I/O and limits performance.

To improve raster image generation performance, AMIGOS allows you to preallocate memory for raster GDV use. This memory can be located in either user or system memory. Whenever AMIGOS needs to allocate memory for raster generation, it first looks for a memory module called RASTER.001. If it cannot find one, it looks for RASTER.002, and so on, up to RASTER.009. If cannot find any such memory modules, it uses the default 8K memory buffer described above.

If it does find a memory module, it allocates it for its use, and uses it for all raster image generation buffering. Note that any such memory modules must be at least 8K in size. The use of multiple memory modules allows for the simultaneous output to multiple raster GDVs, via different GCBs.

The program MAKRST is provided to make it straightforward to allocate these raster memory modules. Use this command followed by the desired memory size to allocate a new raster memory module of the specified size, automatically choosing the appropriate extension.

# CHAPTER 6

# AMIGOS REFERENCE LISTS

This chapter contains quick reference lists for AMIGOS's functions. The first list is alphabetic by function description. The second list is organized into the six major function groups: control functions, output functions, output attributes, input functions, inquiry functions, and mode setting functions; with the function names in each group appearing alphabetically. All of the lists include calling format. For detailed information about the calling format for a given function, please refer to that function's reference sheet in Chapter 7, "Reference Sheets."

## 6.1 ALPHABETIC FUNCTION DESCRIPTION LIST

| FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Bitmap | Assembler | GBM gcb,addr,status |
| memory based | AlphaBASIC | XCALL AMGSBR,G'BMM,gcb,orX,orY,bpp,wid,hgt,pix'arr,flags,status |
| file based | | XCALL AMGSBR,G'BMF,gcb,orX,orY,fil'chanl,flags,status |
| | AlphaC | gbm(gcb,argblock); |
| Clear Workstation | Assembler | GCLRW gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'CLRW,gcb,status |
| | AlphaC | gclrw(gcb); |
| Close Workstation | Assembler | GCLWK gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'CLWK,gcb,status |
| | AlphaC | gclwk(gcb); |
| Escape | Assembler | GESC gcb,argblk,status |
| | AlphaBASIC | XCALL AMGSBR,G'ESC,gcb,argblk,status |
| | AlphaC | gesc(gcb,argblk); |
| Fill Area | Assembler | GFA gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'FA,gcb,point'array,status |
| | AlphaC | gfa(gcb,pointarray); |
| Generalized Drawing Primitive (GDP) | Assembler | GGDP gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'GXXX,gcb,{arguments},status |
| | AlphaC | ggdp(gcb,argblock); |
| GDP - Circle | Assembler | See GDP Reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GCIR,gcb,centx,centY,radius,res,status |
| | AlphaC | See GDP Reference sheet. |
| GDP - Circular Arc | Assembler | See GDP reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GARC,gcb,centX,centY,radius,start,end,res,status |
| | AlphaC | See GDP reference sheet. |
| GDP - Circular Sector | Assembler | See GDP reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GSCT,gcb,centX,centY,radius,start,end,res,status |
| | AlphaC | See GDP reference sheet. |
| GDP - Cubic B-Spline Curve | Assembler | See GDP reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GBCV,gcb,point'array,res,status |
| | AlphaC | See GDP reference sheet. |
| GDP - Ellipse | Assembler | See GDP reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GELI,gcb,centX,centY,radiusX,radiusY,res,stat{,rot} |
| | AlphaC | See GDP reference sheet. |
| GDP - Elliptical Arc | Assembler | See GDP reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GEAR,gcb,centX,centY,radiusX,,radiusY,start,end,res,stat{,rot} |
| | AlphaC | See GDP reference sheet. |

*Alphabetic Function Description List (continued)*

| FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| GDP - Elliptical Sector | Assembler | See GDP reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GESC,gcb,centX,centY,radiusX,radiusY,start,end,res,stat{,rot} |
| | AlphaC | See GDP reference sheet. |
| GDP - Parametric Curve | Assembler | See GDP Reference Sheet |
| | AlphaBASIC | XCALL AMGSBR,G'GPCV,gcb,point'array,res,status |
| | AlphaC | See GDP Reference Sheet |
| GDP - Rectangle | Assembler | See GDP Reference Sheet |
| | AlphaBASIC | XCALL AMGSBR,G'GRCT,gcb,minX,minY,maxX,maxY,status |
| | AlphaC | See GDP Reference Sheet |
| Inquire Color Rep. | Assembler | GQCR gcb,addr,status |
| | ∞ | ∞ |
| HLS mode | AlphaBASIC | XCALL AMGSBR,G'QCR,gcb,index,hue,light,sat,status |
| RGB mode | | XCALL AMGSBR,G'QCR,gcb,index,red,green,blue,status |
| | AlphaC | gqcr(gcb,argblock); |
| Inquire Dynamic Impure Size | Assembler | GQDSZ gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'QDSZ,gcb,name,size,status |
| | AlphaC | gqdsz(gcb); |
| Inquire Error | Assembler | GQERR gcb,error,status |
| | AlphaBASIC | XCALL AMGSBR,G'QERR,gcb,error'mes,status |
| | AlphaC | gqerr(gcb,textptr); |
| Inquire Text Extent | Assembler | GQTXE gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'QTXE,gcb,text,ext'blk,status |
| | AlphaC | gqtxe(gcb,argblock); |
| Inquire Text Representation | Assembly | GQTXR gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'QTXR,gcb,font,ro'type,hgt,rotat,status |
| | AlphaC | gqtxr(gcb,argblock); |
| Inquire Workstation Characteristics | Assembly | GQCHR gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'QCHR,gcb,G'QCHR'MAP,status |
| | AlphaC | gqchr(gcb,argblock); |
| Open Workstation | Assembly | GOPWK gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'OPWK,gcb,name,status,filchnl,term |
| | AlphaC | gopwk(gcb); |
| Polyline | Assembly | GPL gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'PL,gcb,point'array,status |
| | AlphaC | gpl(gcb,pointarray); |
| Polymarker | Assembly | GPM gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'PM,gcb,point'array,status |
| | AlphaC | gpm(gcb,pointarray); |

*Alphabetic Function Description List (continued)*

| FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Request Locator | Assembler | GRQLC gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'RQLC,gcb,ix,iy,rbrbnd,fx,fy,char,val,status |
| | AlphaC | grqlc(gcb,argblock,char); |
| Request Valuator | Assembler | GRQVL gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'RQVL,gcb,value,status |
| | AlphaC | grqvl(gcb,value); |
| Sample Locator | Assembler | GSMLC gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'SMLC,gcb,x,y,char,valid,status |
| | AlphaC | gsmlc(gcb,argblock,char); |
| Sample Valuator | Assembler | GSMVL gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'SMVL,gcb,value,status |
| | AlphaC | gsmvl(gcb,value); |
| Set Character Height | Assembler | GSCHH gcb,height,status |
| | AlphaBASIC | XCALL AMGSBR,G'SCHH,gcb,height,status |
| | AlphaC | gschh(gcb,height); |
| Set Character Rotation | Assembler | GSCHR gcb,rotation,status |
| | AlphaBASIC | XCALL AMGSBR,G'SCHR,gcb,rotation,status |
| | AlphaC | gschr(gcb,rotation); |
| Set Color Mode | Assembler | GSCM gcb,clrmode,status |
| | AlphaBASIC | XCALL AMGSBR,G'SCM,gcb,mode,status |
| | AlphaC | gscm(gcb,colormode); |
| Set Color Rep. | Assembler | GSCR gcb,addr,status |
| HLS mode | AlphaBASIC | XCALL AMGSBR,G'SCR,gcb,clr'idx,hue,light,sat,status |
| RGB mode | | XCALL AMGSBR,G'SCR,gcb,clr'idx,red,green,blue,status |
| | AlphaC | gscr(gcb,argblock); |
| Set Fill Area Color Index | Assembler | GSFAC gcb,color,status |
| | AlphaBASIC | XCALL AMGSBR,G'SFAC,gcb,color,status |
| | AlphaC | gsfac(gcb,fillcolor); |
| Set Fill Area Interior Style | Assembler | GSFAS gcb,style,status |
| | AlphaBASIC | XCALL AMGSBR,G'SFAS,gcb,style,status |
| | AlphaC | gsfas(gcb,fillstyle); |
| Set Fill Area Style Index | Assembler | GSFAI gcb,style,status |
| | AlphaBASIC | XCALL AMGSBR,G'SFAI,gcb,style,status |
| | AlphaC | gsfai(gcb,fillindex); |
| Set Line Type | Assembler | GSPLT gcb,type,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPLT,gcb,type,status |
| | AlphaC | gsplt(gcb,linetype); |
| Set Line Width | Assembler | GSPLS gcb,width,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPLS,gcb,width,status |
| | AlphaC | gspls(gcb,linewidth); |

*Alphabetic Function Description List (continued)*

| FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Set Marker Size | Assembler | GSPMS gcb,size,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPMS,gcb,size,status |
| | AlphaC | gspms(gcb,markersize); |
| Set Marker Type | Assembler | GSPMT gcb,type,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPMT,gcb,type,status |
| | AlphaC | gspmt(gcb,markertype); |
| Set Polyline Color Index | Assembler | GSPLC gcb,color,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPLC,gcb,color,status |
| | AlphaC | gsplc(gcb,linecolor); |
| Set Polymarker Color Index | Assembler | GSPMC gcb,color,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPMC,gcb,color,status |
| | AlphaC | gspmc(gcb,markercolor); |
| Set Text Color Index | Assembler | GSTXC gcb,color,status |
| | AlphaBASIC | XCALL AMGSBR,G'STXC,gcb,color,status |
| | AlphaC | gstxc(gcb,textcolor); |
| Set Text Font | Assembler | GSTXF gcb,font,status |
| | AlphaBASIC | XCALL AMGSBR,G'STXF,gcb,font,status |
| | AlphaC | gstxf(gcb,font); |
| Set Workstation Viewport | Assembler | GSWKV gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'SWKV,gcb,minX,minY,maxX,maxY,status |
| | AlphaC | gswkv(gcb,argblock); |
| Set Workstation Window | Assembler | GSWKW gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'SWKW,gcb,minX,minY,maxX,maxY,status |
| | AlphaC | gswkw(gcb,argblock); |
| Set Writing Mode | Assembler | GSWM gcb,wrtmode,status |
| | AlphaBASIC | XCALL AMGSBR,G'SWM,gcb,mode,status |
| | AlphaC | gswm(gcb,writemode); |
| Text | Assembler | GTX gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'TX,gcb,x,y,string,status |
| | AlphaC | gtx(gcb,argblk); |
| Update Workstation | Assembler | GUPDW gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'UPDW,gcb,status |
| | AlphaC | gupdw(gcb); |

## 6.2 FUNCTION GROUPING LISTS

### Control Function List

| CONTROL FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Clear Workstation | Assembler | GCLRW gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'CLRW,gcb,status |
| | AlphaC | gclrw(gcb); |
| Close Workstation | Assembler | GCLWK gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'CLWK,gcb,status |
| | AlphaC | gclwk(gcb); |
| Escape | Assembler | GESC gcb,argblk,status |
| | AlphaBASIC | XCALL AMGSBR,G'ESC,gcb,argblk,status |
| | AlphaC | gesc(gcb,argblk); |
| Open Workstation | Assembler | GOPWK gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'OPWK,gcb,name,status,filchnl,term |
| | AlphaC | gopwk(gcb); |
| Update Workstation | Assembler | GUPDW gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'UPDW,gcb,status |
| | AlphaC | gupdw(gcb); |

## Output Function List

| OUTPUT FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Bitmap | Assembler | GBM gcb,addr,status |
| Memory based | AlphaBASIC | XCALL AMGSBR,G'BMM,gcb,orX,orY,bpp,wid,hgt,pix'arr,flags,status |
| File based | | XCALL AMGSBR,G'BMF,gcb,orX,orY,fil'chanl,flags,status |
| | AlphaC | gbm(gcb,argblock); |
| Fill Area | Assembler | GFA gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'FA,gcb,point'array,status |
| | AlphaC | gfa(gcb,pointarray); |
| Generalized | Assembler | GGDP gcb,addr,status |
| Drawing | AlphaBASIC | XCALL AMGSBR,G'GXXX,gcb,{arguments},status |
| Primitive (GDP) | AlphaC | ggdp(gcb,argblock); |
| GDP - Circle | Assembler | See GDP Reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GCIR,gcb,centx,centY,radius,res,status |
| | AlphaC | See GDP Reference sheet. |
| GDP - Circular | Assembler | See GDP reference sheet. |
| Arc | AlphaBASIC | XCALL AMGSBR,G'GARC,gcb,centX,centY,radius,start,end,res,status |
| | AlphaC | See GDP reference sheet. |
| GDP - Circular | Assembler | See GDP reference sheet. |
| Sector | AlphaBASIC | XCALL AMGSBR,G'GSCT,gcb,centX,centY,radius,start,end,res,status |
| | AlphaC | See GDP reference sheet. |
| GDP - Cubic | Assembler | See GDP reference sheet. |
| B-Spline | AlphaBASIC | XCALL AMGSBR,G'GBCV,gcb,point'array,res,status |
| Curve | AlphaC | See GDP reference sheet. |
| GDP - Ellipse | Assembler | See GDP reference sheet. |
| | AlphaBASIC | XCALL AMGSBR,G'GELI,gcb,centX,centY,radiusX,radiusY,res,stat{,rot} |
| | AlphaC | See GDP reference sheet. |
| GDP - Elliptical | Assembler | See GDP reference sheet. |
| Arc | AlphaBASIC | XCALL AMGSBR,G'GEAR,gcb,centX,centY,radiusX,,radiusY,start,end,res,stat{,rot} |
| | AlphaC | See GDP reference sheet. |
| GDP - Elliptical | Assembler | See GDP reference sheet. |
| Sector | AlphaBASIC | XCALL AMGSBR,G'GESC,gcb,centX,centY,radiusX,radiusY,start,end,res,stat{,rot} |
| | AlphaC | See GDP reference sheet. |
| GDP - Parametric | Assembler | See GDP Reference Sheet |
| Curve | AlphaBASIC | XCALL AMGSBR,G'GPCV,gcb,point'array,res,status |
| | AlphaC | See GDP reference sheet. |
| GDP - Rectangle | Assembler | See GDP Reference Sheet |
| | AlphaBASIC | XCALL AMGSBR,G'GRCT,gcb,minX,minY,maxX,maxY,status |
| | AlphaC | See GDP reference sheet. |

*Function Grouping List—Output Function (Continued)*

| OUTPUT FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Polyline | Assembler | GPL gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'PL,gcb,point'array,status |
| | AlphaC | gpl(gcb,pointarray); |
| Polymarker | Assembler | GPM gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'PM,gcb,point'array,status |
| | AlphaC | gpm(gcb,pointarray); |
| Text | Assembler | GTX gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'TX,gcb,x,y,string,status |
| | AlphaC | gtx(gcb,argblk); |

## Output Attributes List

| OUTPUT ATTRIBUTES | INTERFACE | CALLING FORMAT |
|---|---|---|
| Set Character Height | Assembler | GSCHH gcb,height,status |
| | AlphaBASIC | XCALL AMGSBR,G'SCHH,gcb,height,status |
| | AlphaC | gschh(gcb,height); |
| Set Character Rotation | Assembler | GSCHR gcb,rotation,status |
| | AlphaBASIC | XCALL AMGSBR,G'SCHR,gcb,rotation,status |
| | AlphaC | gschr(gcb,rotation); |
| Set Color Mode | Assembler | GSCM gcb,clrmode,status |
| | AlphaBASIC | XCALL AMGSBR,G'SCM,gcb,mode,status |
| | AlphaC | gscm(gcb,colormode); |
| Set Color Rep. | Assembler | GSCR gcb,addr,status |
| HLS mode | AlphaBASIC | XCALL AMGSBR,G'SCR,gcb,clr'idx,hue,light,sat,status |
| RGB mode | | XCALL AMGSBR,G'SCR,gcb,clr'idx,red,green,blue,status |
| | AlphaC | gscr(gcb,argblock); |
| Set Fill Area Color Index | Assembler | GSFAC gcb,color,status |
| | AlphaBASIC | XCALL AMGSBR,G'SFAC,gcb,color,status |
| | AlphaC | gsfac(gcb,fillcolor); |
| Set Fill Area Interior Style | Assembler | GSFAS gcb,style,status |
| | AlphaBASIC | XCALL AMGSBR,G'SFAS,gcb,style,status |
| | AlphaC | gsfas(gcb,fillstyle); |
| Set Fill Area Style Index | Assembler | GSFAI gcb,style,status |
| | AlphaBASIC | XCALL AMGSBR,G'SFAI,gcb,style,status |
| | AlphaC | gsfai(gcb,fillindex); |
| Set Line Type | Assembler | GSPLT gcb,type,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPLT,gcb,type,status |
| | AlphaC | gsplt(gcb,linetype); |
| Set Line Width | Assembler | GSPLS gcb,width,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPLS,gcb,width,status |
| | AlphaC | gspls(gcb,linewidth); |
| Set Marker Size | Assembler | GSPMS gcb,size,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPMS,gcb,size,status |
| | AlphaC | gspms(gcb,markersize); |
| Set Marker Type | Assembler | GSPMT gcb,type,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPMT,gcb,type,status |
| | AlphaC | gspmt(gcb,markertype); |
| Set Polyline Color Index | Assembler | GSPLC gcb,color,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPLC,gcb,color,status |
| | AlphaC | gsplc(gcb,linecolor); |

*Function Grouping List—Output Attributes (Continued)*

| OUTPUT ATTRIBUTES | INTERFACE | CALLING FORMAT |
|---|---|---|
| Set Polymarker Color Index | Assembler | GSPMC gcb,color,status |
| | AlphaBASIC | XCALL AMGSBR,G'SPMC,gcb,color,status |
| | AlphaC | gspmc(gcb,markercolor); |
| Set Text Color Index | Assembler | GSTXC gcb,color,status |
| | AlphaBASIC | XCALL AMGSBR,G'STXC,gcb,color,status |
| | AlphaC | gstxc(gcb,textcolor); |
| Set Text Font | Assembler | GSTXF gcb,font,status |
| | AlphaBASIC | XCALL AMGSBR,G'STXF,gcb,font,status |
| | AlphaC | gstxf(gcb,font); |
| Set Workstation Viewport | Assembler | GSWKV gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'SWKV,gcb,minX,minY,maxX,maxY,status |
| | AlphaC | gswkv(gcb,argblock); |
| Set Workstation Window | Assembler | GSWKW gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'SWKW,gcb,minX,minY,maxX,maxY,status |
| | AlphaC | gswkw(gcb,argblock); |
| Set Writing Mode | Assembler | GSWM gcb,wrtmode,status |
| | AlphaBASIC | XCALL AMGSBR,G'SWM,gcb,mode,status |
| | AlphaC | gswm(gcb,writemode); |

## Input Function List

| INPUT FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Request Locator | Assembler | GRQLC gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'RQLC,gcb,ix,iy,rbrbnd,fx,fy,char,val,status |
| | AlphaC | grqlc(gcb,argblock,char); |
| Request Valuator | Assembler | GRQVL gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'RQVL,gcb,value,status |
| | AlphaC | grqvl(gcb,value); |
| Sample Locator | Assembler | GSMLC gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'SMLC,gcb,x,y,char,valid,status |
| | AlphaC | gsmlc(gcb,argblock,char); |
| Sample Valuator | Assembler | GSMVL gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'SMVL,gcb,value,status |
| | AlphaC | gsmvl(gcb,value); |

## Inquiry Function List

| INQUIRY FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Inquire Color Rep. | Assembler | GQCR gcb,addr,status |
| | ∞ | ∞ |
| HLS mode | AlphaBASIC | XCALL AMGSBR,G'QCR,gcb,index,hue,light,sat,status |
| RGB mode | | XCALL AMGSBR,G'QCR,gcb,index,red,green,blue,status |
| | AlphaC | gqcr(gcb,argblock); |
| Inquire Dynamic Impure Size | Assembler | GQDSZ gcb,status |
| | AlphaBASIC | XCALL AMGSBR,G'QDSZ,gcb,name,size,status |
| | AlphaC | gqdsz(gcb); |
| Inquire Error | Assembler | GQERR gcb,error,status |
| | AlphaBASIC | XCALL AMGSBR,G'QERR,gcb,error'mes,status |
| | AlphaC | gqerr(gcb,textptr); |
| Inquire Text Extent | Assembler | GQTXE gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'QTXE,gcb,text,ext'blk,status |
| | AlphaC | gqtxe(gcb,argblock); |
| Inquire Text Representation | Assembler | GQTXR gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'QTXR,gcb,font,ro'type,hgt,rotat,status |
| | AlphaC | gqtxr(gcb,argblock); |
| Inquire Workstation Characteristics | Assembler | GQCHR gcb,addr,status |
| | AlphaBASIC | XCALL AMGSBR,G'QCHR,gcb,G'QCHR'MAP,status |
| | AlphaC | gqchr(gcb,argblock); |

### Mode Setting Function List

| MODE SETTING FUNCTION | INTERFACE | CALLING FORMAT |
|---|---|---|
| Set Writing Mode | Assembler | GSWM gcb,wrtmode,status |
| | AlphaBASIC | XCALL AMGSBR,G'SWM,gcb,mode,status |
| | AlphaC | gswm(gcb,writemode); |

# CHAPTER 7

# REFERENCE SHEETS

This chapter contains reference sheets for AMIGOS's functions which are organized alphabetically by function name. Each sheet shows the function calling information for Assembler, AlphaC and AlphaBASIC languages.

The following table of contents shows the reference sheet name, corresponding generic description and the page number where it appears in this chapter.

# GBM

***Bitmap***

**FUNCTION:**

The GBM function causes an array of color indices to be output to the workstation. Each color index corresponds to a pixel position. The array may reside in a contiguous memory area or in a specially formatted bitmap definition file. (Refer to Appendix C, "BMP Bitmap Image File Format.") The number of bits contained in each color index is variable. The resulting image output shape and appearance are workstation dependent.

**ASSEMBLER CALLING SEQUENCE:**

The Assembler format for the GBM function is:

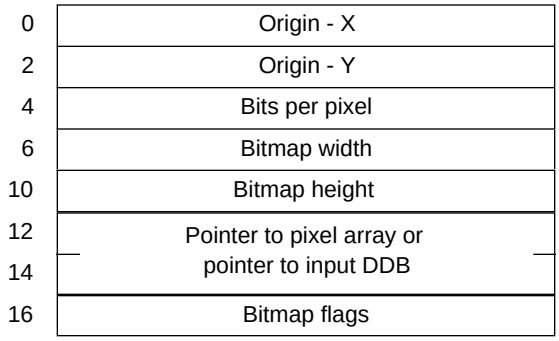
```
GBM     gcb,addr,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described below. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

**Argument Block Format**

The format for the argument block is:




MAC230

**Origin X** and **origin Y** define the starting position of the image in world coordinates. This starting position may be further defined as any of the four corners, lower left, upper left, upper right, or lower right through use of the flags word described below.

The **bits per pixel** value defines the number of bits used to define the color index value for each pixel in the bitmap array. A four color bitmap requires 2 bits per pixel, while an eight color bitmap requires 3 bits per pixel, etc.

The **bitmap width** and **bitmap height** define the size of the entire bitmap array in pixel units.

The **pointer to pixel array** or **pointer to input DDB** field may index a memory based pixel array or a DDB which has been open for file input. The flags word describes the type of input to use.

The **bitmap flags** word contains a number of bits which describe the orientation and source of the bitmap image. The flags are defined in the following table.

| Flag | Definition |
|---|---|

**BM$TRX** **Bitmap Transformation**
Reserved for future use.

**BM$PAK** **Bitmap Packing**
When BM$PAK is set, the bitmap image is defined as conforming to the packing format described in Appendix C, "BMP Bitmap Image File Format." With BM$PAK reset (default) each pixel occupies its defined size in bits per pixel. You do not need to alter this flag if the bitmap source is file oriented.

**BM$FIL** **Bitmap Source is Open File**
When BM$FIL is reset (default) the bitmap image is defined as residing at the memory location defined by the pointer in the argument block.The first location corresponds to the first pixel in the array. The bitmap array is contiguous. Pixels from adjacent rows may reside in the same byte (rows are not byte aligned). When BM$FIL is set, the bitmap image is extracted from a file corresponding to the DDB indexed by the pointer in the argument block. This file must conform to the bitmap file format described in Appendix C, "BMP Bitmap Image File Format." Since the file contains a header describing the size and type of the image, you do not need to include the bits per pixel, width, or height in the call. Additionally, the packing format flag is conditioned by the header information in the file, and does not need to be determined or set by you.

**BM$HOR** **Bitmap Horizontal Origin**
When BM$HOR is reset (default) the horizontal origin is defined as the left side of the image. When BM$HOR is set, the horizontal origin is defined as the right side of the image.

**BM$VOR** **Bitmap Vertical Origin**
When BM$VOR is reset (default) the vertical origin is defined as the bottom of the image. When BM$VOR is set, the vertical origin is defined as the top of the image.

The use of the BM$HOR and BM$VOR flags together produce the following origin locations.

| BM$HOR | BM$VOR | Origin |
|---|---|---|
| 0 | 0 | Bottom Left |
| 0 | 1 | Top Left |
| 1 | 0 | Bottom Right |
| 1 | 1 | Top Right |

| Flag | Definition |
|---|---|
| **BM$VAL** | **Bitmap Vertical Alignment During Clipping**<br>When BM$VAL is reset (default) any clipping will result in the top of the image being visible. When BM$VAL is set, any clipping will result in the bottom of the image being visible. |
| **BM$HAL** | **Bitmap Horizontal Alignment During Clipping**<br>When BM$HAL is reset (default) any clipping will result in the left of the image being visible. When BM$HAL is set, any clipping will result in the right of the image being visible. |

**AlphaC CALLING SEQUENCE:**

The AlphaC format for the GBM function is:

```
gbm(gcb,argblock);   /* bitmap */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `argblock` | Specifies the address of an argument block formatted as described above. |

**Input Parameters:**

```
g_gcb   gcb;                    /* graphics control block */
bmpblk  argblock;               /* bitmap argument block */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* bitmap block */
        {
        gpoint bmporg;          /* x,y coordinates of origin */
        gword bpp;              /* bits per pixel */
        gword width;            /* width in pixels */
        gword height;           /* height in pixels */
        char *arraypnt;         /* pointer to bitmap array
                                   or DDB index */
        gword bmpflg;           /* bitmap flags */
        } bmpblk;
```

### AlphaBASIC CALLING SEQUENCE:

There are two formats for the GBM function in AlphaBASIC. One function defines a memory based bitmap (GBMM), and the other defines a file based bitmap (GBMF).

### Memory Based Bitmap AlphaBASIC Calling Sequence

The format of the GBMM function used to produce an image from an array of pixels residing in memory is:

```
XCALL AMGSBR,G'BMM,gcb,orX,orY,bpp,wid,hgt,pix'arr,flags,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `orX` | This floating point field defines the horizontal origin point in world coordinates. |
| `orY` | This floating point field defines the vertical origin point in world coordinates. |
| `bpp` | This floating point field defines the number of bits per pixel used to specify color index information. |
| `wid` | This floating point field defines the number of horizontal pixels (columns) in the bitmap image. |

`hgt` This floating point field defines the number of vertical pixels (rows) in the bitmap image.

`pix'arr` Specifies an array of pixels which make up the bitmap image. The first pixel corresponds to the upper left corner of the image. Subsequent pixels define the image as progressing horizontally across each row from left to right. The last pixel in the array corresponds to the lower right of the bitmap image. Each pixel consists of a color index made up of the number of bits defined in the bits per pixel field.

`flags` This floating point field defines the flags used for displaying the bitmap image in the proper aspect. The operation of these flags is described above. The file AMGSYM.BSI defines the flags, using the following naming convention:

| | |
|---|---|
| BM'TRX | Reserved for future use. |
| BM'PAK | Bitmap Packing |
| BM'FIL | Bitmap Source is Open File |
| BM'HOR | Bitmap Horizontal Origin |
| BM'VOR | Bitmap Vertical Origin |
| BM'VAL | Bitmap Vertical Alignment During Clipping |
| BM'HAL | Bitmap Horizontal Alignment During Clipping |

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

### File Based Bitmap AlphaBASIC Calling Sequence

The format of the GBMF function used to produce an image from a file conforming to the bitmap file specification defined in Appendix C, "BMP Bitmap Image File Format" is:

```
XCALL AMGSBR,G'BMF,gcb,orX,orY,fil'chanl,flags,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`orX` This floating point field defines the horizontal origin point in world coordinates.

`orY` This floating point field defines the vertical origin point in world coordinates.

`fil'chanl` Specifies an AlphaBASIC file channel which has been open for sequential input. The format of the file to be input on this channel must conform to the bitmap file format defined in Appendix C, "BMP Bitmap Image File Format."

`flags` This floating point field defines the flags used for displaying the bitmap image in the proper aspect. The operation of these flags is described above. The file AMGSYM.BSI defines the flags with the following naming convention:

| | |
|---|---|
| BM'TRX | Bitmap Transformation |
| BM'FIL | Bitmap Source is Open File |
| BM'HOR | Bitmap Horizontal Origin |
| BM'VOR | Bitmap Vertical Origin |
| BM'VAL | Bitmap Vertical Alignment During Clipping |
| BM'HAL | Bitmap Horizontal Alignment During Clipping |

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GCLRW

***Clear Workstation***

**FUNCTION:**

GCLRW initializes a workstation to a known state and clears the display surface.

The results of GCLRW are device dependent. On a display terminal, GCLRW typically clears the screen. On a plotter it loads a new sheet of paper and positions the pen to its "home" position. On a matrix printer, it positions a new sheet of paper into the printing position.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GCLRW function is:

```
GCLRW     gcb,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

**AlphaC CALLING SEQUENCE**

The AlphaC format for the GCLRW function is:

```
gclrw(gcb);   /* clear workstation */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GCLRW function is:

```
XCALL AMGSBR,G'CLRW,gcb,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GCLWK

***Close Workstation***

**FUNCTION:**

The GCLWK function closes a workstation, terminating any operations being performed on that workstation. Before the workstation is closed, any pending operations are completed.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format of the GCLWK function is:

```
GCLWK     gcb,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

**AlphaC CALLING SEQUENCE**

The AlphaC format of the GCLWK function is:

```
gclwk(gcb);   /* close workstation */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format of the GCLWK function is:

```
XCALL AMGSBR,G'CLWK,gcb,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GESC

***Escape***

**FUNCTION:**

The specified non-standard escape function is invoked. The form of the data supplied differs depending on the escape function being invoked.

GESC provides a well-defined method to add extensions to AMIGOS without violating the spirit of the standard.

**ASSEMBLER CALLING SEQUENCE**

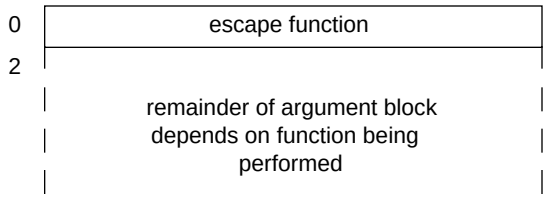
The Assembler format for the GESC function is:

```
GESC    gcb,argblock,status
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`argblock` Specifies the address of an argument block which is passed to the workstation driver for custom processing. The first word of this block contains a function number. Function numbers 0 to 32767 are reserved. Workstation drivers requiring custom processing make use of values 32768 to 65535. The remainder of the argument block is defined by the particular function required by the workstation. If a workstation does not need custom processing, or the function number does not apply to the workstation, no operation is performed.




MAC231

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

### AlphaC CALLING SEQUENCE

The AlphaC format for the GESC function is:

```
gesc(gcb,argblk);   /* escape */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`argblk` Specifies the address of an argument block which is passed to the workstation driver for custom processing. See "Assembler Calling Sequence," above, for details on the format of the argument block.

**Input Parameters:**

```
g_gcb   gcb;                    /* graphics control block */
g_arg   argblk;                 /* workstation dependent block */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct
        {
        gword escfnc;           /* escape function */
        /* remainder of block is workstation dependent */
        } g_arg;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format of the GESC function is:

```
XCALL AMGSBR,G'ESC,gcb,argblock,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`argblock` An unformatted variable containing the function and data to be used by the workstation driver. This area is formatted as described above.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GFA

***Fill Area***

**FUNCTION:**

The GFA function causes the polygon defined by a supplied array to be filled according to the fill area index and style currently selected. The boundary is drawn for interior style hollow and is not drawn for other interior styles. Closure is implied, i.e., the first and last points specified are automatically connected.

Input to this function consists of a number of end points (2-65535) to be connected, followed by a sequence of coordinate pairs describing those end points.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GFA function is:

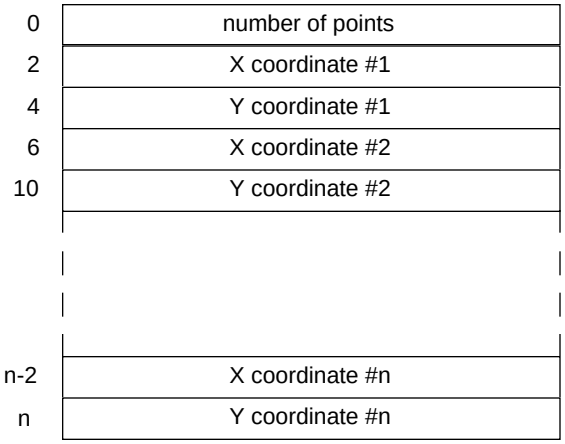
```
GFA     gcb,addr,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described below. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### Argument Block Format

The format of the argument block is:




MAC232

### AlphaC CALLING SEQUENCE

The AlphaC format for the GFA function is:

```
gfa(gcb,pointarray);   /* fill area */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`pointarray` Specifies the address of an argument block formatted as described in "Assembler Calling Sequence," above.

#### Input Parameters:

```
g_gcb   gcb;                    /* graphics control block */
parray  pointarray;             /* array of points */
```

#### Data Types:

```
typedef struct gcb g_gcb;         /* graphics control block */

typedef struct                    /* point array */
        {
        gword pcount;             /* count of active points in array */
        struct gpoint points[n];  /* x and y coordinates */
        } parray;
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GFA function is:

```
XCALL AMGSBR,G'FA,gcb,point'array,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`point'array` Specifies a binary array containing the list of X/Y coordinate pairs defining the area to be displayed. This binary array must be formatted as described in section 3.1.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GGDP

***Generalized Drawing Primitive***

**FUNCTION:**

The GGDP function provides a series of specialized drawing primitives, such as circle, arc, spline generation, etc.

Each of the primitives require a different number and type of arguments.

The supported primitive numbers are:

1 Draw a rectangle using fill area attributes.
2 Draw a circle using fill area attributes.
3 Draw an arc of a circle using polyline attributes.
4 Draw a sector of a circle using fill area attributes.
5 Draw an ellipse using fill area attributes.
6 Draw an arc of an ellipse using polyline attributes.
7 Draw a sector of an ellipse using fill area attributes.
8 Draw a parametric curve using polyline attributes.
9 Draw a cubic B-spline curve using polyline attributes.

Each of the generalized drawing primitive types is described below. Each description includes the argument block layout for the assembly language call, the data structure for the AlphaC call, and the external subroutine call syntax for operation with AlphaBASIC.

In the following argument block definitions, all points are specified in world coordinates. The resolution is specified in tenths of degrees. This determines the number of line segments used to draw the object. Since this number of segments might exceed the maximum number of polygon points allowed, the resolution may not be as accurate as specified. A value of zero causes the best possible resolution to be used.

Starting and ending angles are specified in tenths of degrees with angle 0 extending in a positive direction along the X axis. Angles increase in a counterclockwise direction.

## ASSEMBLER CALLING SEQUENCE

The Assembler format of the GGDP function is:

```
GGDP    gcb,addr,status
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`addr` Specifies the address of an argument block formatted as described below.

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

## AlphaC CALLING SEQUENCE

Unlike the Assembler calling sequence, each generalized drawing primitive uses a separate function in AlphaC. Each function consists of a GCB index, a variable number of arguments, and an optional returned status. The general format is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`argblock` Specifies the address of an argument block formatted as described below.

**AlphaBASIC CALLING SEQUENCE**

Unlike the Assembler calling sequence, each generalized drawing primitive uses a separate function in AlphaBASIC. Each function consists of a GCB index, a variable number of arguments, and an optional returned status. The general format is:

```
XCALL AMGSBR,G'GXXX,gcb,{arguments},status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`arguments` This is a variable list of parameters supplied to define the primitive being drawn. All arguments are floating point with the exception of the point arrays supplied to the parametric and cubic B-spline curve primitives. These point arrays are formatted as described in section 3.1. The AlphaBASIC program must supply all arguments. There are no optional or default arguments in this function.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

***GDP - Circle***

A circle is drawn using fill area attributes by specifying the center point and radius.

The argument block format for the circle function is:

| | |
|---|---|
| 0 | G$GCIR (2) |
| 2 | Center X |
| 4 | Center Y |
| 6 | Radius |
| 10 | Resolution |

MAC233

## AlphaC CALLING SEQUENCE

The AlphaC format for a circle is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

### Input Parameters:

```
g_gcb   gcb;                    /* graphics control block */
circle  argblock;               /* generalized drawing primitive
                                   argument block. */
```

### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* circle descriptor block */
        {
        gword gdptyp = 2;       /* gdp type code for circle */
        gpoint center;          /* center point coordinates */
        gword radius;           /* radius */
        gword resolution;       /* resolution in tenth degrees */
        } circle;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGCIR is:

```
XCALL AMGSBR,G'GCIR,gcb,centX,centY,radius,res,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `centX` | A floating point value which defines the center X point of the circle. |
| `centY` | A floating point value which defines the center Y point of the circle. |
| `radius` | A floating point value which defines the radius of the circle. |
| `res` | A floating point value which defines the resolution of the circle. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

***GDP - Circular Arc***

A circular arc is drawn using polyline attributes by specifying the center point, radius, starting angle, and ending angle.

The argument block format for the circular arc function is:

| | |
|---|---|
| 0 | G$GARC (3) |
| 2 | Center X |
| 4 | Center Y |
| 6 | Radius |
| 10 | Starting angle |
| 12 | Ending Angle |
| 14 | Resolution |

MAC234

## AlphaC CALLING SEQUENCE

The AlphaC format for a circular arc is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

### Input Parameters:

```
g_gcb   gcb;                    /* graphics control block */
circarc argblock;               /* generalized drawing primitive
                                   argument block. */
```

### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* circular arc descriptor block */
        {
        gword gdptyp = 3;       /* gdp type code for circular arc */
        gpoint center;          /* center point coordinates */
        gword radius;           /* radius */
        gword startangle;       /* starting angle in tenth degrees */
        gword endangle;         /* ending angle in tenth degrees */
        gword resolution;       /* resolution in tenth degrees */
        } circarc;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGARC is:

```
XCALL AMGSBR,G'GARC,gcb,centX,centY,radius,start,end,res,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `centX` | A floating point value which defines the center X point of the arc. |
| `centY` | A floating point value which defines the center Y point of the arc. |
| `radius` | A floating point value which defines the radius of the arc. |
| `start` | A floating point value which defines the starting angle of the arc. |
| `end` | A floating point value which defines the ending angle of the arc. |
| `res` | A floating point value which defines the resolution of the arc. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### GDP - Circular Sector

A circular sector (pie wedge) is drawn using fill area attributes by specifying the center point, radius, starting angle, and ending angle.

The argument block format for the circular sector function is:

| | |
|---|---|
| 0 | G$GSCT (4) |
| 2 | Center X |
| 4 | Center Y |
| 6 | Radius |
| 10 | Starting angle |
| 12 | Ending Angle |
| 14 | Resolution |

MAC235

## AlphaC CALLING SEQUENCE

The AlphaC format for a circular sector is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

**Input Parameters:**

```
g_gcb   gcb;                    /* graphics control block */
circsec argblock;               /* generalized drawing primitive
                                   argument block. */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* circular sector descriptor block */
        {
        gword gdptyp = 4;       /* gdp type code for circular sector */
        gpoint center;          /* center point coordinates */
        gword radius;           /* radius */
        gword startangle;       /* starting angle in tenth degrees */
        gword endangle;         /* ending angle in tenth degrees */
        gword resolution;       /* resolution in tenth degrees */
        } circsec;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGSCT is:

```
XCALL AMGSBR,G'GSCT,gcb,centX,centY,radius,start,end,res,status
```

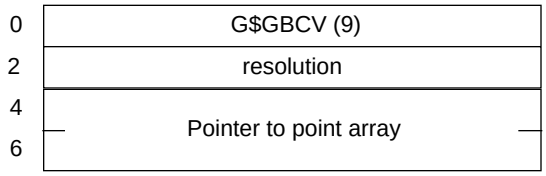
where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `centX` | A floating point value defining the X center point of the sector. |
| `centY` | A floating point value defining the Y center point of the sector. |
| `radius` | A floating point value defining the radius of the sector. |
| `start` | A floating point value defining the starting angle of the sector. |
| `end` | A floating point value defining the ending angle of the sector. |
| `res` | A floating point value defining the resolution of the sector. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## GDP - Cubic B-spline Curve

A smooth cubic B-spline curve is drawn to connect the points defined by a point array. This curve approaches the points defined in the array. By adding duplicate data points, the curve will more closely track the data points.

The argument block format for the cubic B-spline curve function is:




MAC236

### AlphaC CALLING SEQUENCE

The AlphaC format for a cubic B-spline curve is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

**Input Parameters:**

```
g_gcb gcb;                      /* graphics control block */
b_spline argblock;              /* generalized drawing primitive
                                   argument block. */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* B-spline descriptor block */
        {
        gword gdptyp = 9;       /* gdp type code for B-spline */
        gword resolution;       /* resolution in number of segments
                                   between adjacent points */
        char *pointarray;       /* pointer to point array */
        } b_spline;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGBCV is:

```
XCALL AMGSBR,G'GBCV,gcb,point'array,res,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `point' array` | Specifies a binary array containing the list of X/Y coordinate pairs defining the curve to be displayed. This binary array must be formatted as described in section 3.1. |
| `res` | Floating point value defining resolution. Expressed as number of line segments to draw between contiguous points. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

***GDP - Ellipse***

An ellipse is drawn using fill area attributes by specifying the center point, radius in X direction, and radius in Y direction.

The argument block format for the ellipse function is:

| | |
|---|---|
| 0 | G$GELI (5) |
| 2 | Center X |
| 4 | Center Y |
| 6 | Radius X |
| 10 | Radius Y |
| 12 | Resolution |
| 14 | Rotation |

MAC237

## AlphaC CALLING SEQUENCE

The AlphaC format for an ellipse is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

### Input Parameters:

```
g_gcb gcb;                      /* graphics control block */
ellipse argblock;               /* generalized drawing primitive
                                   argument block. */
```

### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* ellipse descriptor block */
        {
        gword gdptyp = 5;       /* gdp type code for ellipse */
        gpoint center;          /* center point coordinates */
        gword radiusx;          /* radius in x direction */
        gword radiusy;          /* radius in y direction */
        gword resolution;       /* resolution in tenth degrees */
        gword rotation;         /* rotation in tenth degrees */
        } ellipse;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGELI is:

```
XCALL AMGSBR,G'GELI,gcb,centX,centY,radiusX,radiusY,res,stat{,rotation}
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `centX` | A floating point value defining the X center point of the ellipse. |
| `centY` | A floating point value defining the Y center point of the ellipse. |
| `radiusX` | A floating point value defining the X direction radius of the ellipse. |
| `radiusY` | A floating point value defining the Y direction radius of the ellipse. |
| `res` | A floating point value defining the resolution of the ellipse. |
| `stat` | Specifies a register in which 16-bits of status are returned. Appendix A, "Status Codes and Messages" gives status field value definitions. |
| `rotation` | An optional floating point value specifying the rotation of the ellipse expressed in tenths of degrees. |

## GDP - Elliptical Arc

An elliptical arc is drawn using polyline attributes by specifying the center point, radius in X direction, radius in Y direction, starting angle, and ending angle.

The argument block format of the elliptical arc function is:

| | |
|---|---|
| 0 | G$GEAR (6) |
| 2 | Center X |
| 4 | Center Y |
| 6 | Radius X |
| 10 | Radius Y |
| 12 | Starting angle |
| 14 | Ending angle |
| 16 | Resolution |
| 20 | Rotation |

MAC238

### AlphaC CALLING SEQUENCE

The AlphaC format for an elliptical arc is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

#### Input Parameters:

```
g_gcb gcb;                      /* graphics control block */
elliparc argblock;              /* generalized drawing primitive
                                   argument block. */
```

#### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* elliptical arc descriptor block */
        {
        gword gdptyp = 6;       /* gdp type code for elliptical arc */
        gpoint center;          /* center point coordinates */
        gword radiusx;          /* radius in x direction  */
        gword radiusy;          /* radius in y direction */
        gword startangle;       /* starting angle in tenth degrees */
        gword endangle;         /* ending angle in tenth degrees */
        gword resolution;       /* resolution in tenth degrees */
        gword rotation;         /* rotation in tenth degrees */
        } elliparc;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGEAR is:

```
XCALL AMGSBR,G'GEAR,gcb,centX,centY,radX,radY,start,end,res,stat{,rotation}
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `centX` | A floating point value defining the X center point of the arc. |
| `centY` | A floating point value defining the Y center point of the arc. |
| `radX` | A floating point value defining the X direction radius of the arc. |
| `radY` | A floating point value defining the Y direction radius of the arc. |
| `start` | A floating point value defining the starting angle of the arc. |
| `end` | A floating point value defining the ending angle of the arc. |
| `res` | A floating point value defining the resolution of the arc. |
| `stat` | Specifies a register in which 16-bits of status are returned. Appendix A, "Status Codes and Messages" gives status field value definitions. |
| `rotation` | An optional floating point number specifying the rotation of the elliptical arc expressed in tenths of degrees. |

## GDP - Elliptical Sector

An elliptical sector is drawn using fill area attributes by specifying the center point, radius in X direction, radius in Y direction, starting angle, and ending angle.

The argument block format for the elliptical sector function is:

| | |
|---|---|
| 0 | G$GESC (7) |
| 2 | Center X |
| 4 | Center Y |
| 6 | Radius X |
| 10 | Radius Y |
| 12 | Starting angle |
| 14 | Ending angle |
| 16 | Resolution |
| 20 | Rotation |

MAC239

### AlphaC CALLING SEQUENCE

The AlphaC format for an elliptical sector is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

**Input Parameters:**

```
g_gcb gcb;                      /* graphics control block */
ellipsec argblock;              /* generalized drawing primitive
                                   argument block. */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* elliptical sector descriptor block */
        {
        gword gdptyp = 7;       /* gdp type code for elliptical sector */
        gpoint center;          /* center point coordinates */
        gword radiusx;          /* radius in x direction  */
        gword radiusy;          /* radius in y direction */
        gword startangle;       /* starting angle in tenth degrees */
        gword endangle;         /* ending angle in tenth degrees */
        gword resolution;       /* resolution in tenth degrees */
        gword rotation;         /* rotation in tenth degrees */
        } ellipsec;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGESC is:

```
XCALL AMGSBR,G'GESC,gcb,centX,centY,radX,radY,start,end,res,stat{,rotation}
```

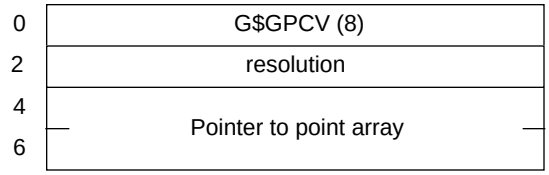
where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `centX` | Floating point value defining the X center point of the sector. |
| `centY` | Floating point value defining the Y center point of the sector. |
| `radX` | Floating point value defining the X direction radius of the sector. |
| `radY` | Floating point value defining the Y direction radius of the sector. |
| `start` | Floating point value defining the starting angle of the sector. |
| `end` | Floating point value defining the ending angle of the sector. |
| `res` | Floating point value defining the resolution of the sector. |
| `stat` | Specifies a register in which 16-bits of status are returned. Appendix A, "Status Codes and Messages" gives status field value definitions. |
| `rotation` | An optional floating point number specifying the rotation of the elliptical sector expressed in tenths of degrees. |

***GDP - Parametric Curve***

A smooth parametric curve is drawn to connect the points defined by a point array. This curve passes through the points.

The argument block format for the parametric curve function is:




MAC240

## AlphaC CALLING SEQUENCE

The AlphaC format for a parametric curve is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

### Input Parameters:

```
g_gcb gcb;                      /* graphics control block */
parcurve argblock;              /* generalized drawing primitive
                                   argument block. */
```

### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* parametric curve descriptor block */
        {
        gword gdptyp = 8;       /* gdp type code for parametric curve */
        gword resolution;       /* resolution in number of segments
                                   between adjacent points */
        char *pointarray;       /* pointer to point array */
        } parcurve;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGPCV is:

```
XCALL AMGSBR,G'GPCV,gcb,point'array,res,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `point' array` | Specifies a binary array containing the list of X/Y coordinate pairs defining the curve to be displayed. This binary array must be formatted as described in section 3.1. |
| `res` | Floating point value defining resolution. Expressed as number of line segments to draw between contiguous points. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## GDP - Rectangle

A rectangle is drawn using fill area attributes by specifying the lower left and upper right coordinate points.

The argument block format for the rectangle function is:

| | |
|---|---|
| 0 | G$GRCT (1) |
| 2 | Minimum X |
| 4 | Minimum Y |
| 6 | Maximum X |
| 10 | Maximum Y |

MAC241

### AlphaC CALLING SEQUENCE

The AlphaC format for a rectangle is:

```
ggdp(gcb,argblock);   /* generalized drawing primitive */
```

**Input Parameters:**

```
g_gcb gcb;                      /* graphics control block */
rectangle argblock;             /* generalized drawing primitive
                                   argument block. */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* rectangle descriptor block */
        {
        gword gdptyp = 1;       /* gdp type code for rectangle */
        gpoint corner1;         /* first corner coordinates */
        gpoint corner2;         /* second corner coordinates */
        } rectangle;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for GGRCT is:

```
XCALL AMGSBR,G'GRCT,gcb,minX,minY,maxX,maxY,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `minX` | A floating point value which defines the lower left X point of the rectangle. |
| `minY` | A floating point value which defines the lower left Y point of the rectangle. |
| `maxX` | A floating point value which defines the upper right X point of the rectangle. |
| `maxY` | A floating point value which defines the upper right Y point of the rectangle. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GOPWK

***Open Workstation***

**FUNCTION:**

GOPWK selects a given workstation for input and output operations by intializing it. You must open a workstation with GOPWK before any operations can be performed with it.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GOPWK function is:

```
GOPWK     gcb,status
```

where:

`gcb` Specifies the address of a graphics control block. This block must have the workstation name field (GC.NAM) filled in prior to the GOPWK function. If this field is null—contains zeros—then your terminal will be used as the default workstation, providing that the terminal is a graphics workstation.

Each workstation type defaults to a specific output type. For example, the TK4105.GDV assumes all output is routed to the user terminal, while LASWRT.GDV by default opens the file LASPLT.PS for output. You can specify an alternate output file by placing the index to an output DDB in the field GC.OUT prior to the GOPWK function. This DDB must be initialized and open for output. You must close this file after use. The field GC.OUT must be zero if you do not want to use this feature.

You can also specify an alternate output terminal. By placing the name of the terminal, as defined by TRMDEF during system initialization, in the GC.TNM variable in the GCB, all graphics output will be routed through the terminal output system to the specified terminal. The name must be packed in RAD50 format and be placed in GC.TNM prior to the GOPWK function.

If the dynamic impure area pointer (GC.DPT) field is zero, AMIGOS will allocate any necessary dynamic impure space through the use of the GETMEM monitor call through AMOS. Alternately, you can set this field to point to an impure area previously allocated. The required size of this area may be determined by the inquire dynamic impure size function (GQDSZ).

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

### AlphaC CALLING SEQUENCE

The AlphaC format for the GOPWK function is:

```
gopwk(gcb);     /* open workstation */
```

For information on the gcb argument, refer to the discussion above in "Assembler Calling Sequence."

#### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
```

#### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GOPWK function is:

```
XCALL AMGSBR,G'OPWK,gcb,name,status,filchnl,term
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. The size of this area must be large enough to contain the GCB and any additional dynamic impure areas required by AMIGOS and the workstation driver. |
| `name` | Specifies a string giving the name of the workstation which is to be opened. If this field is null, the user terminal will be used as the graphics device. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |
| `filchnl` | Specifies an optional file channel number for an output file which has been open for output. All output will be routed to this file channel if this parameter is specified. |
| `term` | Specifies an optional terminal name to route all graphics output to. This string variable must match a terminal defined in a system initialization file TRMDEF command. |

# GPL

***Polyline***

**FUNCTION:**

GPL generates a sequence of connected straight lines, starting from the first point and ending at the last point.

The input to GPL consists of a number of end points (2-65535) to be connected, followed by a sequence of coordinate pairs describing those end points.

If the polyline width exceeds the maximum width supported by the workstation, the lines will be generated by AMIGOS as filled areas, using the current polyline attributes. The intersection points of the line segments will be mitered with a filled circle using a radius equal to one half the polyline width.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GPL function is:

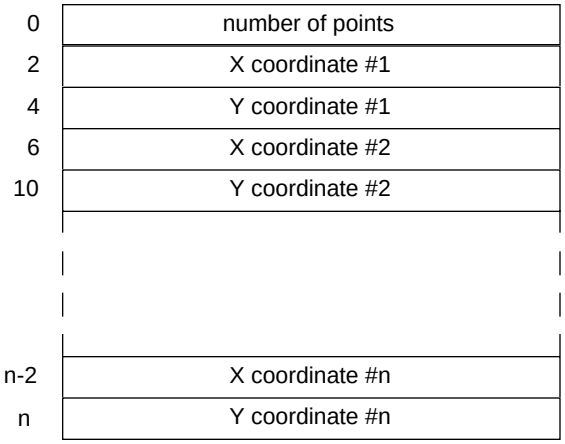
```
GPL     gcb,addr,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described below. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### Argument Block Format

The format of the argument block is:




MAC232

### AlphaC CALLING SEQUENCE

The AlphaC format for the GPL function is:

```
gpl(gcb,pointarray);     /* polyline */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`pointarray` Specifies the address of an argument block formatted as described below.

#### Input Parameters:

```
g_gcb gcb;                      /* graphics control block */
parray pointarray;              /* array of points */
```

#### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* point array */
        {
        gword pcount;           /* count of active points in array */
        struct gpoint points[n]; /* x and y coordinates */
        } parray;
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GPL function is:

```
XCALL AMGSBR,G'PL,gcb,point'array,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `point' array` | Specifies a binary array containing the list of X/Y coordinate pairs defining the list of lines to be displayed. This binary array must be formatted as described in section 3.1. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GPM

***Polymarker***

**FUNCTION:**

The GPM function generates a sequence of markers to identify all of the given positions.

The input to GPM consists of a number of points (1-65535) where markers are to be placed, followed by a sequence of coordinate pairs describing those points.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GPM function is:

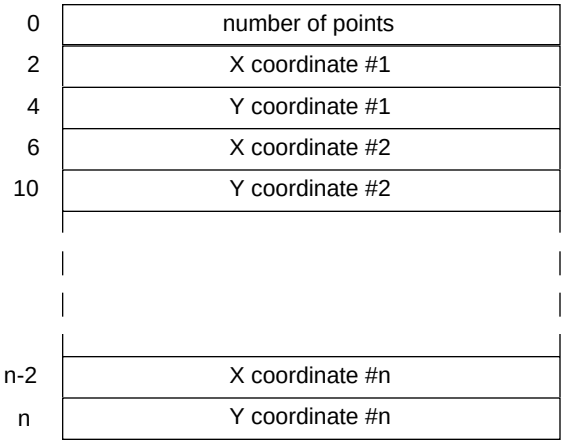
```
GPM     gcb,addr,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described below. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### Argument Block Format

The format of the argument block is:




MAC232

### AlphaC CALLING SEQUENCE

The AlphaC format for the GPM function is:

```
gpm(gcb,pointarray);     /* polymarker */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`pointarray` Specifies the address of an argument block formatted as described below.

#### Input Parameters:

```
g_gcb   gcb;                    /* graphics control block */
parray  pointarray;             /* array of points */
```

#### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* point array */
        {
        gword pcount;           /* count of active points in array */
        struct gpoint points[n]; /* x and y coordinates */
        } parray;
```

**AlphaBASIC CALLING SEQUENCE**

The format of the AlphaBASIC GPM function is:

```
XCALL AMGSBR,G'PM,gcb,point'array,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `point' array` | Specifies a binary array containing the list of X/Y coordinate pairs defining the list of markers to be displayed. This binary array must be formatted as described in section 3.1. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GQCHR

***Inquire Workstation Characteristics***

**FUNCTION:**

The GQCHR function returns the workstation characteristics block from the graphics device driver (.GDV)

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GQCHR function is:

```
GQCHR    gcb,addr,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described in the file AMGSYM.M68. The size of this area should be equal or larger than GH.SIZ. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GQCHR function is:

```
gqchr(gcb,argblock);     /* inquire workstation characteristics */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`argblock` Specifies the address of an argument block formatted as described in the file AMIGOS.H.

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
```

### Output Parameters:

```
chrblk *argblock;             /* characteristics block */
```

### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct wschar chrblk;   /* characteristics block */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GQCHR function is:

```
XCALL AMGSBR,G'QCHR,gcb,G'QCHR'MAP,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`G'QCHR'MAP` Specifies a workstation characteristics area formatted as defined in the file AMGSYM.BSI.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GQCR

***Inquire Color Representation***

**FUNCTION:**

GQCR returns the current color representation for a specified index. The color representation may be returned as HLS (Hue, Lightness, Saturation) or RGB (red, green, blue), depending on the current color mode.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GQCR function is:

```
GQCR    gcb,addr,status
```

where:

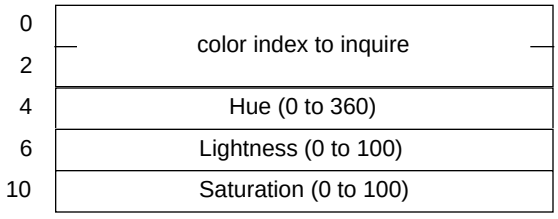
`gcb` Specifies the address of the graphics control block associated with the workstation used.

`addr` Specifies the address of an argument block formatted as described below.

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

**HLS Color Mode Argument Block Format**

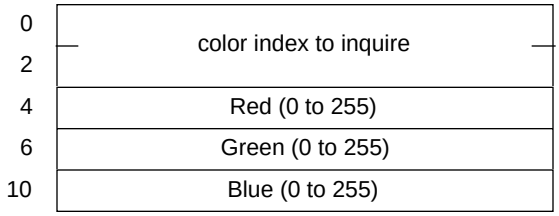
The format of the argument block in HLS color mode is:




MAC243

### RGB Color Mode Argument Block Format

The format of the argument block in RGB color mode is:




MAC244

### AlphaC CALLING SEQUENCE

The AlphaC format for the GQCR function is:

```
gqcr(gcb,argblock);     /* inquire color representation */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `argblock` | Specifies the address of an argument block formatted as described below. |

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
```

**Input/Output Parameters:**

```
clrblk argblock;              /* color representation argument */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef struct                /* color representation argument */
        {
        glong index;          /* color index to modify */
        gword hue_red;        /* hue or red
        gword lgt_green;      /* lightness or green */
        gword sat_blue;       /* saturation or blue */
        } clrblk;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GQCR function in HLS color mode:

```
XCALL AMGSBR,G'QCR,gcb,index,hue,light,sat,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `index` | A variable containing the color index number to inquire. |
| `hue` | A floating point variable to receive the Hue in the range 0 - 360. |
| `light` | A variable to receive the lightness in the range 0 - 100. |
| `sat` | A variable to receive the saturation in the range 0 - 100. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

The AlphaBASIC format for the GQCR function in RGB color mode:

```
XCALL AMGSBR,G'QCR,gcb,index,red,green,blue,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `index` | A variable containing the color index number to inquire. |
| `red` | A floating point variable to receive the red component of the color representation. |
| `green` | A floating point variable to receive the green component of the color representation. |
| `blue` | A floating point variable to receive the blue component of the color representation. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GQDSZ

***Inquire Dynamic Impure Size***

**FUNCTION:**

The GQDSZ function determines the amount of impure memory space required to successfully open a workstation for subsequent use.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GQDSZ function is:

```
GQDSZ     gcb,status
```

where:

`gcb` Specifies the address of a graphics control block. This block must have the workstation name field (GC.NAM) set in the same manner as the open workstation (GOPWK) function. Refer to the GOPWK function for more information. If GC.NAM is null (zeroes), AMIGOS assumes the user's terminal will be used as a workstation in a subsequent open workstation function. AMIGOS defaults to a maximum of 500 points per polygon generated through use of the fill area (GFA) function. The user may specify a larger number of points by placing the desired count in the number of polygon output points (GC.OPP) field. The impure size required is returned in the dynamic impure size (GC.DSZ) field. The user may use this size to allocate an impure zone prior to the open workstation (GOPWK) function. The pointer to this impure area must be put in the dynamic impure pointer (GC.DPT) field prior the the GOPWK function.

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

### AlphaC CALLING SEQUENCE

The AlphaC format for the GQDSZ function is:

```
gqdsz(gcb);     /* inquire dynamic impure size */
```

where:

`gcb` Specifies the address of a graphics control block. See "Assembler Calling Sequence" above for details on this argument.

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GQDSZ function is:

```
XCALL AMGSBR,G'QDSZ,gcb,name,size,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`name` Specifies a string giving the name of the workstation which is to be opened. If this field is null, the user terminal will be used as the graphics device.

`size` Specifies a floating point variable in which the required size of the dynamic impure area is returned.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GQERR

***Inquire Error***

**FUNCTION:**

GQERR causes the pointer to an error message corresponding to the status in GC.ERR to be returned. The error message is terminated with a null byte.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GQERR function is:

```
GQERR     gcb,error,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `error` | Specifies a register in which the pointer the ASCII error message associated with the current error condition is returned. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### AlphaC CALLING SEQUENCE

The AlphaC format for the GQERR function is:

```
gqerr(gcb,textptr);     /* inquire error */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `textptr` | Specifies a register in which the pointer the ASCII error message associated with the current error condition is returned. |

**Input Parameters:**

```
g_gcb gcb;                  /* graphics control block */
```

**Output Parameters:**

```
glong *textptr;             /* pointer to error string */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef unsigned glong;         /* 4-byte integer */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GQERR function is:

```
XCALL AMGSBR,G'QERR,gcb,error'mes,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `error'mes` | A string or unformatted variable to receive the error message. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GQTXE

***Inquire Text Extent***

**FUNCTION:**

The GQTXE function returns the coordinates of a rectangle corresponding to the size of a supplied text string. This rectangle represents the current text attributes, including height, font, and rotation. The returned coordinate points are expressed in world coordinates. Since the text string may extend beyond the current viewport boundaries, these coordinates are returned as longword (32-bit) signed integers. The rectangle is positioned relative to the lower left corner of the first character such that the first coordinate pair is always 0,0. The remaining coordinate pairs follow in a sequence which follows a counterclockwise path around the bounding rectangle.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GQTXE function is:

```
GQTXE    gcb,addr,status
```

where:

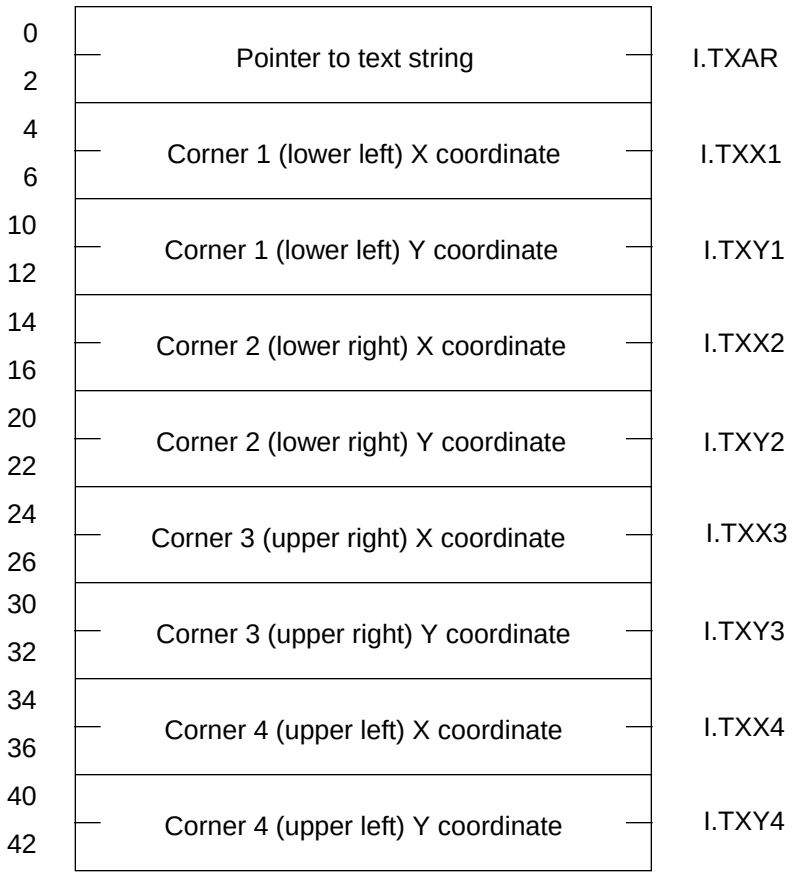
`gcb` Specifies the address of the graphics control block associated with the workstation used.

`addr` Specifies the address of an argument block formatted as described below. Only the text string pointer needs to be supplied when making the call. The text string must be terminated with a null (0) byte.

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

**Argument Block Format**

The format of the argument block is:




MAC245

### AlphaC CALLING SEQUENCE

The AlphaC format for the GQTXE function is:

```
gqtxe(gcb,argblock);     /* inquire text extent */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`argblock` Specifies the address of an argument block formatted as described below. Only the text string pointer needs to be supplied when making the call. The text string must be terminated with a null (0) byte.

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
```

**Input/Output Parameters:**

```
extblk argblock;              /* text extent argument */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* text extent argument block */
        {
        char   *itxar;          /* pointer to text string */
        glong  itxx1;           /* lower left x */
        glong  itxy1;           /* lower left y */
        glong  itxx2;           /* lower right x */
        glong  itxy2;           /* lower right y */
        glong  itxx3;           /* upper right x */
        glong  itxy3;           /* upper right y */
        glong  itxx4;           /* upper left x */
        glong  itxy4;           /* upper left y */
        } extblk;
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format of the GQTXE function is:

```
XCALL AMGSBR,G'QTXE,gcb,text,ext'blk,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`text` A string variable which contains the text for which the extent is requested.

`ext'blk` This argument specifies a text extent rectangle formatted as follows:

```
MAP1 EXTENT'BLOCK
   MAP2 EXTENT'CORNER(4)
      MAP3 EXTENT'X,F
      MAP3 EXTENT'Y,F
```

In the above block, each of four corners contains an X and Y coordinate. The corners proceed in a counterclockwise fashion. Corner 1 is lower left, corner 2 is lower right, corner 3 is upper right, and corner 4 is upper left. These locations apply to a rectangle with no rotation.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GQTXR

***Inquire Text Representation***

**FUNCTION:**

The GQTXR function returns the current text attributes being used by the workstation. Since the workstation need not exactly match the specified text attributes, this function provides a means to determine the attributes which will actually be used to produce the text.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GQTXR function is:

```
GQTXR     gcb,addr,status
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`addr` Specifies the address of an argument block formatted as described below. The workstation GDV returns the current text attributes in effect to this block.

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

**Argument Block Format**

The format of the argument block is:

| | | |
|---|---|---|
| 0 | Text font being used | I.TRFN |
| 2 | Rotation type supported | I.TRRT |
| 4 | Actual character height | I.TRCH |
| 6 | Actual character rotation | I.TRCR |

MAC246

All returned values are 16 bit. The rotation type supported defines the capability of the workstation to rotate characters. The types are as follows:

| | |
|---|---|
| I$TRNR | Workstation does not support rotation. |
| I$TR90 | Workstation supports 90 degree rotation. |
| I$TR45 | Workstation supports 45 degree rotation. |
| I$TRFR | Workstation provides full rotation. |

The text font, actual character height, and actual character rotation define the current attributes in use by the workstation. These attributes may be used to determine actual text positioning and appearance.

## AlphaC CALLING SEQUENCE

The AlphaC format for the GQTXR function is:

```
gqtxr(gcb,argblock);   /* inquire text representation */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`argblock` Specifies the address of an argument block formatted as described above. The workstation GDV returns the current text attributes in effect to this block.

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
```

### Output Parameters:

```
textblk argblock;             /* text representation argument */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef struct                /* text representation argument block */
        {
        gword font;           /* font being used */
        gword rotatetype;     /* rotation type supported */
        glong height;         /* actual character height */
        glong rotation;       /* actual character rotation */
        } textblk;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GQTXR function is:

```
XCALL AMGSBR,G'QTXR,gcb,font,ro'type,hght,rotation,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`font` This floating point variable receives the current text font number.

`ro' type` This floating point variable receives the type of rotation supported by the workstation. The rotation types are:

| | |
|---|---|
| 0 | Workstation does not support rotation. |
| 1 | Workstation supports 90 degree rotation. |
| 2 | Workstation supports 45 degree rotation. |
| 3 | Workstation supports full rotation. |

`hght` This floating point variable receives the current text height which will be used by the workstation for subsequent text output.

`rotation` This floating point variable receives the current text rotation which will be used by the workstation for subsequent text output.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GRQLC

***Request Locator***

**FUNCTION:**

The GRQLC function causes the specified workstation to return a pair of world coordinates corresponding to a workstation dependent locator. This may be realized by a number of input methods: mouse, joystick, joydisk, cursor keys, digitizing tablet, etc. The coordinates are returned in a user defined argument block. Certain workstations may provide an additional character input corresponding to a particular keystroke or button depression. This character is returned in register D1. If bit 31 of D1 is set, the operator has selected a location outside of the current viewport and the coordinates returned may not be accurate. This is an interactive function which requires the operator to perform a workstation dependent function (such as pressing a key or mouse button) before the coordinates are returned. Some workstations support rubberbanding. This will appear as a moving line or rectangle on the display until the operator actuates a key or button. The type of rubberbanding desired is defined in the function. Not all workstations support this feature and will ignore associated fields in the argument block.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GRQLC function is:

```
GRQLC    gcb,addr,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described below. |
| `status` | Specifies a register in which 32 bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

**Argument Block Format**

The format of the argument block is as follows:

| | |
|---|---|
| 0 | Initial X coordinate |
| 2 | Initial Y coordinate |
| 4 | Rubberband type |
| 6 | Final X coordinate |
| 10 | Final Y coordinate |

MAC247

The initial X and Y coordinates are used to specify the starting position of the cursor or the anchor point of a rubberband function. The rubberband types are defined as follows:

| | |
|---|---|
| RB$OFF | No rubberbanding. |
| RB$LIN | Rubberband line. |
| RB$RCT | Rubberband rectangle. |

The final X and Y coordinates are returned in the argument block at offsets 6 and 10.

## AlphaC CALLING SEQUENCE

The AlphaC format for the GRQLC function is:

```
grqlc(gcb,argblock,chr);     /* request locator */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `argblock` | Specifies the address of an argument block formatted as described below. |
| `chr` | Specifies a pointer to a string variable to receive the returned character from the pointing device. |

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
```

### Input/Output Parameters:

```
rqlblk argblock;              /* request locator argument block */
```

### Output Parameters:

```
char chr[1];                  /* string to receive character */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef struct                /* request locator */
        {
        gpoint initpos;       /* initial coordinate point */
        gword rbandtype;      /* rubber band type */
        gpoint finalpos;      /* final returned position */
        } rqlblk;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GRQLC function is:

```
XCALL AMGSBR,G'RQLC,gcb,ix,iy,rbrbnd,fx,fy,char,val,stat
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`ix` This floating point field specifies the initial X coordinate to be used as a rubberband anchor.

`iy` This floating point field specifies the initial Y coordinate to be used as a rubberband anchor.

`rbrbnd` This floating point field specifies the type of rubberbanding to be used. The rubber band types are:

| | |
|---|---|
| 0 | No rubberbanding. |
| 1 | Rubberband line. |
| 2 | Rubberband rectangle. |

`fx` This floating point field receives the final X coordinate.

`fy` This floating point field receives the final Y coordinate.

`char` This string variable receives the character corresponding to a key press or button depression.

`val` This floating point field represents the validity of the returned coordinates. If this field is zero, the coordinates are valid. If this field is non-zero, the operator has selected a location outside of the current viewport and the final coordinates may be erroneous.

`stat` Specifies the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GRQVL

***Request Valuator***

**FUNCTION:**

The GRQVL function requesting valuator returns an integer value in a workstation dependent manner. This may correspond to a scalar value or potentiometer input. This is an interactive function and requires you to perform a workstation dependent function to return the value. The integer value is returned in register D1.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GRQVL function is:

```
GRQVL     gcb,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `status` | Specifies a register in which 32 bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GRQVL function is:

```
grqvl(gcb,value);     /* request valuator */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `value` | Specifies a pointer to a variable to receive the final value. |

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
```

**Output Parameters:**

```
glong *value;                 /* requested value */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GRQVL function is:

```
XCALL AMGSBR, G'RQVL,gcb,value,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `value` | This floating point field receives the final value. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSCHH

***Set Character Height***

**FUNCTION:**

GSCHH sets the height of all characters output by subsequent text functions. Depending on the selected font, the workstation may map the character height to the closest height available, rather than use the exact value specified.

Character height is specified in world coordinates. Most fonts will have a capital height (height of upper case alphabet character) of approximately 73 percent of this total height. In most cases, this allows sufficient room for character ascenders and descenders when setting lines solid—that is, with no additional space between lines other than the text height itself.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSCHH function is:

```
GSCHH    gcb,height,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `height` | Specifies the height of the characters, in world coordinates, to be output in subsequent text operations. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSCHH function is:

```
gschh(gcb,height);     /* set character height */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`height` Specifies the height of the characters, in world coordinates, to be output in subsequent text operations.

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
glong height;                 /* character height */
```

### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef unsigned glong;         /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSCHH function is:

```
XCALL AMGSBR,G'SCHH,gcb,height,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`height` This floating point field specifies the height of the characters to be output in subsequent text operations.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GSCHR

***Set Character Rotation***

**FUNCTION:**

The specified character rotation is stored for use when generating subsequent text output primitives.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSCHR function is:

```
GSCHR    gcb,rotation,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `rotation` | Specifies the rotation angle in tenths of degrees. The rotation angle is specified as a counterclockwise rotation from the positive X axis. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### AlphaC CALLING SEQUENCE

The AlphaC format for the GSCHR function is:

```
gschr(gcb,rotation);   /* set character height */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`rotation` Specifies the rotation angle in tenths of degrees. The rotation angle is specified as a counterclockwise rotation from the positive X axis.

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
glong rotation;               /* rotation in tenth degrees */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format of the GSCHR function is:

```
XCALL AMGSBR,G'SCHR,gcb,rotation,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`rotation` A floating point field which defines the rotation angle in tenths of degrees. The rotation angle is specified as a counterclockwise rotation from the positive X axis.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GSCM

***Set Color Mode***

**FUNCTION:**

GSCM sets the color definition mode for subsequent GSCR (set color representation) and GQCR (inquire color representation) functions. The mode may be set to either the HLS (hue, lightness, saturation) or RGB (red, green, blue) mode.

**ASSEMBLER CALLING SEQUENCE**

The format for the GSCM function is:

```
GSCM    gcb,clrmode,status
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`clrmode` Specifies the color mode to be used for subsequent color setting and inquiry functions. The defined color modes are as follows:

0 = HLS (hue, lightness, saturation)
1 = RGB (red, green, blue)

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSCM function is:

```
gscm(gcb,colormode);     /* set color mode */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`colormode` Specifies the color mode to be used for subsequent color setting and inquiry functions. The defined color modes are as follows:

0 = HLS (hue, lightness, saturation)
1 = RGB (red, green, blue)

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
glong colormode;              /* color mode */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSCM function is:

```
XCALL AMGSBR,G'SCM,gcb,mode,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`mode` A floating point field defining the color mode to be used for subsequent output operations. The allowable values for this field are defined above.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GSCR

***Set Color Representation***

**FUNCTION:**

In the color table of a given workstation, each color index is associated with a specific color. The selected color is then mapped to the closest available value available on the workstation.

The workstation color table has predefined entries based on the characteristics of the workstation; at least indices 0 and 1 are predefined for every workstation. Any table entry, including the predefined ones, may be redefined using this function.

When any output primitive (line, marker, text, etc.) is displayed, the color index refers to an entry in the color table. If output primitives are displayed with a color index that is not present in the color table, a workstation dependent color will be used. The background color is defined by color index 0.

GSCR sets the current color representation for a specified index. The color representation may be returned as HLS (Hue, Lightness, Saturation) or RGB (red, green, blue), depending on the current color mode.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSCR function is:

```
GSCR    gcb,addr,status
```

where:

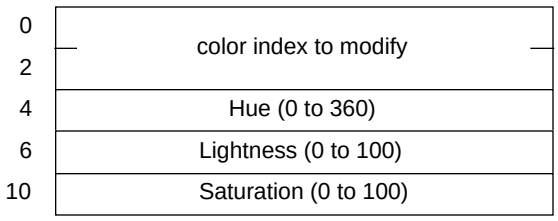
`gcb` Specifies the address of the graphics control block associated with the workstation used.

`addr` Specifies the address of an argument block formatted as described below.

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

### HLS Mode Argument Block Format

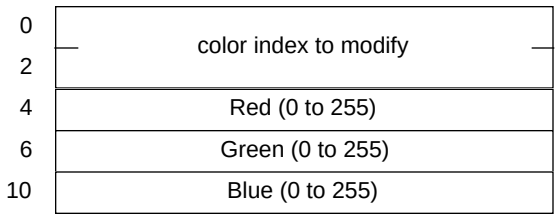
The format of the argument block in HLS mode is:




MAC248

### RGB Mode Argument Block Format

The format of the argument block in RGB mode is:




MAC249

### AlphaC CALLING SEQUENCE

The AlphaC format for the GSCR function is:

```
gscr(gcb,argblock);     /* set color representation */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described below. |

**Input Parameters:**

```
g_gcb gcb;                  /* graphics control block */
clrblk argblock;            /* color representation argument */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* color representation argument */
        {
        glong index;            /* color index to modify */
        gword hue_red;          /* hue or red
        gword lgt_green;        /* lightness or green */
        gword sat_blue;         /* saturation or blue */
        } clrblk;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GSCR function in HLS mode is:

```
XCALL AMGSBR,G'SCR,gcb,clr'idx,hue,light,sat,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `clr'idx` | A variable containing the number of the color index to get. |
| `hue` | A variable containing the hue component in range 0-360. |
| `light` | A variable containing the lightness component in range 0-100. |
| `sat` | A variable containing the saturation component in range 0-100. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

The AlphaBASIC format for the GSCR function in RGB mode is:

```
XCALL AMGSBR,G'SCR,gcb,clr'idx,red,green,blue,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `clr'idx` | A variable containing the number of the color index to get. |
| `red` | A variable containing the red component in range 0-255. |
| `blue` | A variable containing the blue component in range 0-255. |
| `green` | A variable containing the green component in range 0-255. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSFAC

***Set Fill Area Color Index***

**FUNCTION:**

GSFAC determines the color index to be used by subsequent fill area operations. If a workstation does not support color, this function is ignored. The number of possible colors which may be selected is workstation dependent.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSFAC function is:

```
GSFAC     gcb,color,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `color` | Specifies the color index to be used in subsequent fill area operations. This field is an index into the workstation's color representation table. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSFAC function is:

```
gsfac(gcb,fillcolor);     /* set fill area color */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`fillcolor` Specifies the color index to be used in subsequent fill area operations. This field is an index into the workstation's color representation table.

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
glong fillcolor;              /* fill area color */
```

### Data Types:

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef unsigned glong;         /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSFAC function is:

```
XCALL AMGSBR,G'SFAC,gcb,color,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`color` A floating point field defining the color index to be used in subsequent fill area operations. This field is an index into the workstation's color index table.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GSFAI

***Set Fill Area Style Index***

**FUNCTION:**

GSFAI sets the fill area style to be used in subsequent fill area operations. For the hollow and solid interior styles, this setting is ignored. For the pattern interior style, it specifies which of the workstation dependent patterns to use.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSFAI function is:

```
GSFAI    gcb,index,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `index` | Specifies pattern style to be used. This argument is an index into the workstation's style table. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSFAI function is:

```
gsfai(gcb,fillindex);     /* set fill area index */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`fillindex` Specifies pattern style to be used. This argument is an index into the workstation's style table.

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
glong fillindex;              /* fill area style index */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSFAI function is:

```
XCALL AMGSBR,G'SFAI,gcb,index,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`index` This floating point field specifies pattern style to be used. This argument is an index into the workstation's style table.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GSFAS

***Set Fill Area Internal Style***

**FUNCTION:**

GSFAS sets the type of interior fill to be used in subsequent filled areas.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSFAS function is:

```
GSFAS    gcb,style,status
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`style` Specifies the interior style to be used in subsequent fill area operations. The defined interior styles are:

1 = hollow
2 = solid
3 = pattern
4 = hatch
>4 = device dependent

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

**AlphaC CALLING SEQUENCE**

The AlphaC format for the GSFAS function is:

```
gsfas(gcb,fillstyle);     /* set fill area style */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`fillstyle` Specifies the interior style to be used in subsequent fill area operations. The defined interior styles are:

1 = hollow
2 = solid
3 = pattern
4 = hatch
>4 = device dependent

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
glong fillstyle;              /* fill area style */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSFAS function is:

```
XCALL AMGSBR,G'SFAS,gcb,style,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `style` | A floating point field defining the interior style to be used in subsequent fill area functions. The allowable values for this field are defined above. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSMLC

***Sample Locator***

**FUNCTION:**

The GSMLC function causes the specified workstation to return a pair of world coordinates corresponding to a workstation dependent locator. This may be realized by a number of input methods, such as: mouse, joystick, joydisk, cursor keys, digitizing tablet, etc.

The coordinates are returned in a user defined argument block. This is a non-interactive input function which returns the current locator without operator intervention. Certain workstations may return a character corresponding to a key or button that was pressed prior to the function. This character is returned in register D1. If bit 31 of D1 is set, the current location is outside of the current viewport and the coordinates returned may not be accurate.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSMLC function is:

```
GSMLC    gcb,addr,status
```

where:

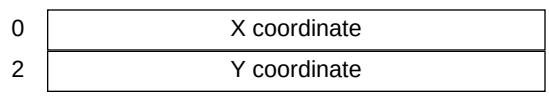
`gcb` Specifies the address of the graphics control block associated with the workstation used.

`addr` Specifies the address of an argument block formatted as described below.

`status` Specifies a register in which 32 bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

**Argument Block Format**

The format of the argument block is as follows:




MAC250

### AlphaC CALLING SEQUENCE

The AlphaC format for the GSMLC function is:

```
gsmlc(gcb,argblock,chr);     /* sample locator */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `argblock` | Specifies the address of an argument block formatted as described below. |
| `chr` | Specifies a pointer to a string variable to receive the returned character from the pointing device. |

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
```

**Output Parameters:**

```
smlblk argblock;              /* sample locator argument block */
char chr[1];                  /* string to receive character */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef struct gpoint smlblk; /* returned locator point */
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GSMLC function is:

```
XCALL AMGSBR,G'SMLC,gcb,x,y,char,valid,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `x` | This floating point field receives the current X coordinate. |
| `y` | This floating point field receives the current Y coordinate. |
| `char` | This string variable receives the character corresponding to the pressing of a key or button which may have occurred prior to the call. |
| `valid` | This floating point field represents the validity of the returned coordinates. If this field is zero, the coordinates are valid. If this field is non-zero, the location is outside of the current viewport and the final coordinates may be erroneous. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSMVL

***Sample Valuator***

**FUNCTION:**

The GSMVL function returns an integer value in a workstation dependent manner. This may correspond to a scalar value or potentiometer input. This is an non-interactive function and does not require operator intervention in order to return a value. The integer value is returned in register D1.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSMVL function is:

```
GSMVL     gcb,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `status` | Specifies a register in which 32 bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSMVL function is:

```
gsmvl(gcb,value);     /* sample valuator */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `value` | Specifies a pointer to a variable to receive the final value. |

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
```

### Output Parameters:

```
glong *value;                 /* sampled value */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSMVL function is:

```
XCALL AMGSBR,G'SMVL,gcb,value,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `value` | This floating point field receives the final value. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSPLC

***Set Polyline Color Index***

**FUNCTION:**

GSPLC determines the color index to be used by subsequent polyline operations. If a workstation does not support color, this function is ignored. The number of possible colors which may be selected is workstation dependent.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSPLC function is:

```
GSPLC     gcb,color,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `color` | Specifies the color index to be used in subsequent polyline operations. This field is an index into the workstation's color representation table. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### AlphaC CALLING SEQUENCE

The AlphaC format for the GSPLC function is:

```
gsplc(gcb,linecolor);     /* set line color */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`linecolor` Specifies the color index to be used in subsequent polyline operations. This field is an index into the workstation's color representation table.

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
glong linecolor;              /* character height */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSPLC function is:

```
XCALL AMGSBR,G'SPLC,gcb,color,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`color` A floating point field defining the color index to be used in subsequent polyline operations. This field is an index into the workstation's color index table.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GSPLS

***Set Line Width***

**FUNCTION:**

GSPLS sets the width of the lines to be drawn via subsequent polyline operations. Line width is specified in world coordinates. The line width produced is device dependent. Each device supports a nominal line width which may be used by setting this value to zero. If the specified line width exceeds that which the workstation can produce, solid lines are drawn using the fill area function. All other line types are matched to the workstation capabilities as closely as possible.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSPLS function is:

```
GSPLS    gcb,width,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `width` | Defines the line width in world coordinates. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSPLS function is:

```
gspls(gcb,linewidth);     /* set line width */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `linewidth` | Defines the line width in world coordinates. |

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
glong linewidth;              /* line width */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSPLS function is:

```
XCALL AMGSBR,G'SPLS,gcb,width,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `width` | A floating point field defining the line width. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSPLT

***Set Line Type***

**FUNCTION:**

GSPLT sets the line type to be used in subsequent polyline operations.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSPLT function is:

```
GSPLT     gcb,type,status
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`type` Specifies the line type to be used in subsequent polyline operations. The defined line types are as follows:

1 = solid line
2 = dashed line
3 = dotted line
>3 = device dependent

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

**AlphaC CALLING SEQUENCE**

The AlphaC format for the GSPLT function is:

```
gsplt(gcb,linetype);     /* set line type */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`linetype` Specifies the line type to be used in subsequent polyline operations. The defined line types are as follows:

1 = solid line
2 = dashed line
3 = dotted line
>3 = device dependent

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
glong linetype;               /* line type */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSPLT function is:

```
XCALL AMGSBR,G'SPLT,gcb,type,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `type` | A floating point field defining the line type to be used. The allowable values for this field are defined above. |
| `status` | Specifies a floating point variable in which the the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSPMC

***Set Polymarker Color Index***

**FUNCTION:**

GSPMC determines the color index to be used by subsequent polymarker operations. If a workstation does not support color, this function is ignored. The number of possible colors which may be selected is workstation dependent.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSPMC function is:

```
GSPMC     gcb,color,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `color` | Specifies the color index to be used in subsequent polymarker operations. This field is an index into the workstation's color representation table. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSPMC function is:

```
     gspmc(gcb,markercolor);                                    /*  set
polymarker color */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`markercolor` Specifies the color index to be used in subsequent polymarker operations. This field is an index into the workstation's color representation table.

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
glong markercolor;            /* polymarker color */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSPMC function is:

```
     XCALL AMGSBR,G'SPMC,gcb,color,status
```

where:

`gcb` Specifies an unformatted (type X) variable which is the graphics control block for the workstation used.

`color` A floating point field defining the color index to be used in subsequent polyline operations. This field is an index into the workstation's color index table.

`status` Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages."

# GSPMS

***Set Marker Size***

**FUNCTION:**

GSPMS sets the size of the markers to be drawn via subsequent polymarker operations. The marker size is related to the nominal marker size on a workstation; the result is mapped by the workstation to the nearest available marker size.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSPMS function is:

```
GSPMS    gcb,size,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `width` | Defines the marker size in world coordinates. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### AlphaC CALLING SEQUENCE

The AlphaC format for the GSPMS function is:

```
gspms(gcb,markersize);     /* set polymarker size */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `markersize` | Defines the marker size in world coordinates. |

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
glong markersize;             /* polymarker size */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSPMS function is:

```
XCALL AMGSBR,G'SPMS,gcb,size,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `scale` | A floating point field defining the size of the markers to be used in subsequent polymarker operations. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSPMT

***Set Marker Type***

**FUNCTION:**

GSPMT sets the marker type to be used in subsequent polymarker operations.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSPMT function is:

```
GSPMT     gcb,type,status
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`type` Specifies the marker type to be used in subsequent polymarker operations. The defined marker types are as follows:

| | |
|---|---|
| 1 = dot (.) | 4 = circle (O) |
| 2 = plus (+) | 5 = cross (X) |
| 3 = star (*) | >5 = device dependent |

`status` Specifies a register in which 16-bits of status are returned. Appendix A, "Status Codes and Messages" gives status field value definitions.

### AlphaC CALLING SEQUENCE

The AlphaC format for the GSPMT function is:

```
gspmt(gcb,markertype);      /* set polymarker type */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`markertype` Specifies the marker type to be used in subsequent polymarker operations. The defined marker types are as follows:

| | |
|---|---|
| 1 = dot (.) | 4 = circle (O) |
| 2 = plus (+) | 5 = cross (X) |
| 3 = star (*) | >5 = device dependent |

**Input Parameters:**

```
g_gcb gcb;                  /* graphics control block */
glong markertype;           /* marker type */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef unsigned glong;         /* 4-byte integer */
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GSPMT function is:

```
XCALL AMGSBR,G'SPMT,gcb,type,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `type` | A floating point field defining the marker type to be used in subsequent polymarker operations. Allowable values are defined above. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Appendix A, "Status Codes and Messages" gives status field value definitions. |

# GSTXC

***Set Text Color Index***

**FUNCTION:**

GSTXC determines the color index to be used by subsequent text operations. If a workstation does not support color, this function is ignored. The number of possible colors which may be selected is workstation dependent.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSTXC function is:

```
GSTXC     gcb,color,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `color` | Specifies the color index to be used in subsequent text operations. This field is an index into the workstation's color representation table. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSTXC function is:

```
gstxc(gcb,textcolor);     /* set text color */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `textcolor` | Specifies the color index to be used in subsequent text operations. This field is an index into the workstation's color representation table. |

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
glong textcolor;              /* text color */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSTXC function is:

```
XCALL AMGSBR,G'STXC,gcb,color,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `color` | A floating point field defining the color index to be used in subsequent text operations. This field is an index into the workstation's color index table. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSTXF

***Set Text Font***

**FUNCTION:**

GSTXF determines the font to be used by subsequent text functions.

If the specified text font is not available on a workstation, font 1 is used. If a font above 1001 is specified and not found in memory, font 1001, Simplex Roman, is used.

All workstations which support graphics output are guaranteed to have font 1 as a minimum.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSTXF function is:

```
GSTXF    gcb,font,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `font` | Specifies the font to be used for subsequent text operations. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

## AlphaC CALLING SEQUENCE

The AlphaC format for the GSTXF function is:

```
gstxf(gcb,font);     /* set text font */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `font` | Specifies the font to be used for subsequent text operations. |

### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
glong font;                   /* text font */
```

### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

## AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSTXF function is:

```
XCALL AMGSBR,G'STXF,gcb,font,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `font` | A floating point variable which specifies the font to be used in subsequent text operations. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSWKV

***Set Workstation Viewport***

**FUNCTION:**

GSWKV allows display of the currently windowed portion of the world coordinate space in a portion of the entire display area. In this way, multiple representations of the world space can reside in a single display area.

To set a viewport, the desired lower left and upper right coordinates of the display space are specified. This results in all subsequent output operations being reduced or enlarged to fill the specified display area. When a workstation is opened, the workstation viewport is set equal to the entire world coordinate space.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSWKV function is:

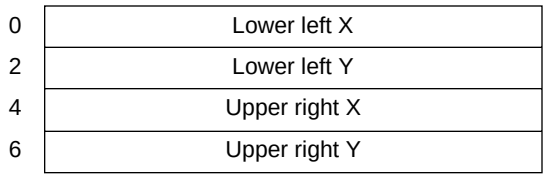
```
GSWKV     gcb,addr,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described below. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

### Argument Block Format

The format of the argument block is:




MAC228

### AlphaC CALLING SEQUENCE

The AlphaC format for the GSWKV function is:

```
gswkv(gcb,argblock);     /* set viewport */
```

where:

`gcb` Specifies the address of the graphics control block associated with the workstation used.

`argblock` Specifies the address of an argument block formatted as described below.

#### Input Parameters:

```
g_gcb gcb;                    /* graphics control block */
viewblk argblock;             /* viewport argument */
```

#### Data Types:

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef struct                /* viewport argument block */
        {
        gpoint corner1;       /* first corner of viewport */
        gpoint corner2;       /* second corner of viewport */
        } viewblk;
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSWKV function is:

```
XCALL AMGSBR,G'SWKV,gcb,minX,minY,maxX,maxY,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `minX` | This floating point value defines the lower left X coordinate of the desired viewport. |
| `minY` | This floating point value defines the lower left Y coordinate of the desired viewport. |
| `maxX` | This floating point value defines the upper right X coordinate of the desired viewport. |
| `maxY` | This floating point value defines the upper right Y coordinate of the desired viewport. |
| `status` | This field specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSWKW

***Set Workstation Window***

**FUNCTION:**

GSWKW allows you to view only a portion of the entire world coordinate space. This portion of the full space is displayed in the currently active display area (viewport) of the workstation.

To set a window, the desired lower left and upper right coordinates of the world space are specified. This results in all subsequent output operations being reduced or enlarged to fill the current viewport. Output primitives which lie outside of the window area are not displayed. When a workstation is opened, the workstation window is set equal to the entire world coordinate space.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSWKW function is:

```
GSWKW     gcb,addr,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `addr` | Specifies the address of an argument block formatted as described below. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

**Argument Block Format**

The format of the argument block is:

| | |
|---|---|
| 0 | Lower left X |
| 2 | Lower left Y |
| 4 | Upper right X |
| 6 | Upper right Y |

MAC228

### AlphaC CALLING SEQUENCE

The AlphaC format for the GSWKW function is:

```
gswkw(gcb,argblock);      /* set window */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `argblock` | Specifies the address of an argument block formatted as described below. |

**Input Parameters:**

```
g_gcb gcb;                      /* graphics control block */
winblk argblock;                /* window argument */
```

**Data Types:**

```
typedef struct gcb g_gcb;       /* graphics control block */

typedef struct                  /* viewport argument block */
        {
        gpoint corner1;         /* first corner of window */
        gpoint corner2;         /* second corner of window */
        } winblk;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GSWKW function is:

```
XCALL AMGSBR,G'SWKW,gcb,minX,minY,maxX,maxY,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `minX` | This floating point value defines the lower left X coordinate of the desired window |
| `minY` | This floating point value defines the lower left Y coordinate of the desired window |
| `maxX` | This floating point value defines the upper right X coordinate of the desired window |
| `maxY` | This floating point value defines the upper right Y coordinate of the desired window |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GSWM

***Set Writing Mode***

**FUNCTION:**

GSWM sets the writing mode for the workstation for subsequent output operations. Not all workstations support all writing modes. Certain workstation types support writing modes only in pixel operations during bitmap output. GSWM's function is workstation dependent.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GSWM function is:

```
GSWM    gcb,wrtmode,status
```

where:

`gcb` Specifies the graphics control block address associated with the workstation used.

`wrtmode` Specifies the writing mode to be used for subsequent output operations. The defined writing modes are as follows:

0 = replace mode
1 = exclusive or mode (XOR)
2 = logical and mode (AND)
3 = logical or mode (OR)
>3 = device dependent

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

**AlphaC CALLING SEQUENCE**

The AlphaC format for the GSWM function is:

```
gswm(gcb,writemode);     /* set writing mode */
```

where:

`gcb` Specifies the graphics control block address associated with the workstation used.

`writemode` Specifies the writing mode to be used for subsequent output operations. The defined writing modes are as follows:

0 = replace mode
1 = exclusive or mode (XOR)
2 = logical and mode (AND)
3 = logical or mode (OR)
>3 = device dependent

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
glong writemode;              /* writing mode */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */

typedef unsigned glong;       /* 4-byte integer */
```

### AlphaBASIC CALLING SEQUENCE

The AlphaBASIC format for the GSWM function is:

```
XCALL AMGSBR,G'SWM,gcb,mode,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `mode` | A floating point field defining the writing mode to be used for subsequent output operations. The allowable values for this field are defined above. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GTX

***Text***

**FUNCTION:**

The GTX function causes the display of a character string. The current values of the text attributes are used. The starting position is specified within the function. Text rotation is applied to text string relative to the origin point specified within the function. If the workstation is not able to produce the exact attributes, the closest approximation is used. The user may call the GQTXR routine to determine the actual attributes which will be used.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GTX function is:

```
GTX     gcb,addr,status
```

where:

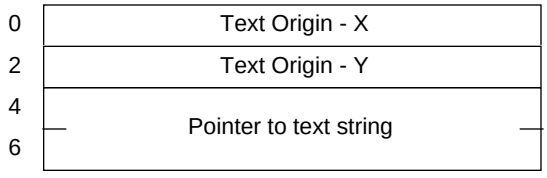
`gcb` Specifies the address of the graphics control block associated with the workstation used.

`addr` Specifies the address of a memory block formatted as described below. The text string must be terminated by a null (0) byte.

`status` Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages."

**Argument Block Format**

The format of the text argument block is:




MAC242

### AlphaC CALLING SEQUENCE

The AlphaC format for the GTX function is:

```
gtx(gcb,argblk);     /* text */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `argblk` | Specifies the address of a memory block formatted as described below. The text string must be terminated by a null (0) byte. |

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
txtblk argblk;                /* text argument block */
```

**Data Types:**

```
typedef struct gcb g_gcb;        /* graphics control block */

typedef struct                   /* text argument block */
        {
        gpoint origin;           /* x,y coordinates of origin */
        char *text_pnt;          /* pointer to text string */
        } txtblk;
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GTX function is:

```
XCALL AMGSBR,G'TX,gcb,x,y,string,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `x` | A floating point field specifying the X-axis component of the coordinate at which to display the text string. |
| `y` | A floating point field specifying the Y-axis component of the coordinate at which to display the text string. |
| `string` | A string field containing the text to be displayed. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# GUPDW

***Update Workstation***

**FUNCTION:**

GUPDW causes any deferred actions currently pending for the workstation to be executed. If no deferred actions are pending, GUPDW performs no operation. If an alternate output path is active through the GC.OUT field, any buffered data is output at this time.

**ASSEMBLER CALLING SEQUENCE**

The Assembler format for the GUPDW function is:

```
GUPDW     gcb,status
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |
| `status` | Specifies a register in which 16-bits of status are returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

**AlphaC CALLING SEQUENCE**

The AlphaC format for the GUPDW function is:

```
gupdw(gcb);     /* update workstation */
```

where:

| | |
|---|---|
| `gcb` | Specifies the address of the graphics control block associated with the workstation used. |

**Input Parameters:**

```
g_gcb gcb;                    /* graphics control block */
```

**Data Types:**

```
typedef struct gcb g_gcb;     /* graphics control block */
```

**AlphaBASIC CALLING SEQUENCE**

The AlphaBASIC format for the GUPDW function is:

```
XCALL AMGSBR,G'UPDW,gcb,status
```

where:

| | |
|---|---|
| `gcb` | Specifies an unformatted (type X) variable which is the graphics control block for the workstation used. |
| `status` | Specifies a floating point variable in which the status of the complete operation is returned. Status field values are defined in Appendix A, "Status Codes and Messages." |

# APPENDIX A

# STATUS CODES AND MESSAGES

AMIGOS returns the following decimal codes and corresponding messages after a call.

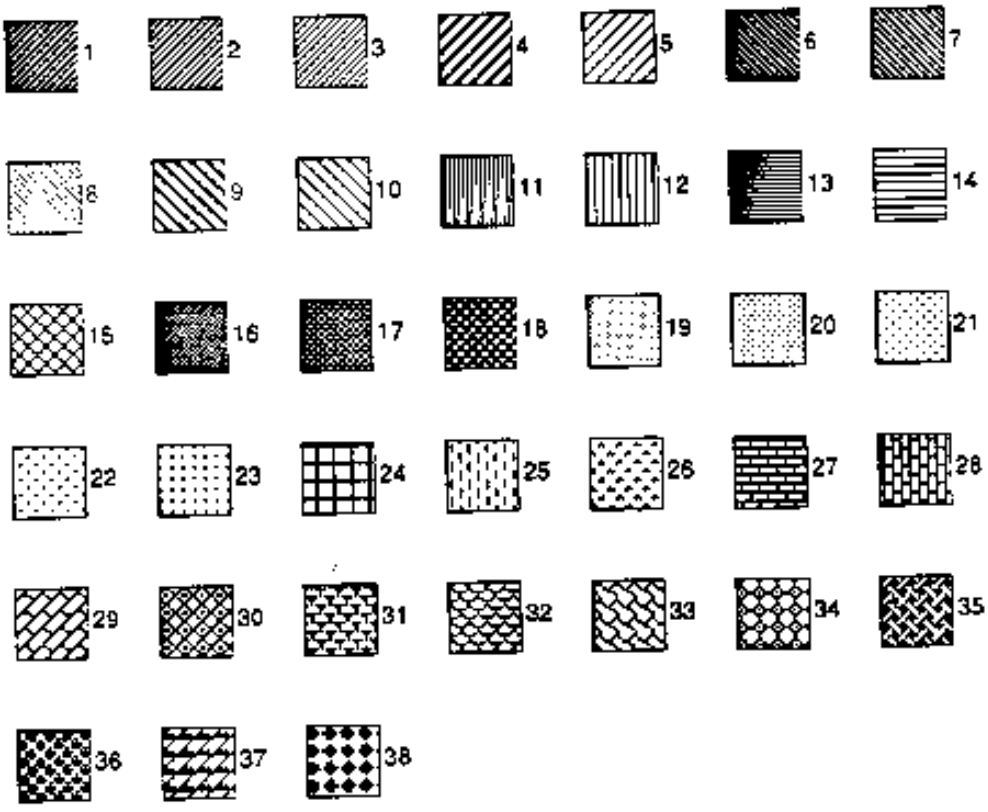
| CODE | SYMBOL | MESSAGE |
|---|---|---|
| 0-255 | | Standard AMOS file system errors. |
| 256-511 | | Unused by AMIGOS. |
| 512 | G$EWSA | Workstation is active. |
| 513 | G$ENPR | Workstation has no pixel readback. |
| 514 | G$EGDP | Invalid GDP specified in argument. |
| 515 | G$EANP | AMIGOS not present in system. |
| 516 | G$ENEP | Not enough data points provided. |
| 517 | G$EWNO | Workstation not open. |
| 518 | G$ENWK | Unable to locate workstation. |
| 519 | G$EILT | Invalid line type. |
| 520 | G$EILW | Invalid line width. |
| 521 | G$EICI | Invalid color index. |
| 522 | G$EIMT | Invalid marker type. |
| 523 | G$EIFS | Invalid fill area interior style. |
| 524 | G$EIFI | Invalid fill area index. |
| 525 | G$ETNF | Alternate terminal in GC.TNM not found. |
| 526 | G$EOUT | Workstation does not support output. |
| 527 | G$EINP | Workstation does not support input. |
| 528 | G$EEOF | Unexpected end of bitmap file. |
| 529 | G$EMEM | Insufficient memory for dynamic area. |
| 530 | G$EWIN | Improper window specification. |
| 531 | G$EVEW | Improper viewport specification. |
| 532 | G$EFNO | File channel not open (AMGSBR). |
| 533 | G$EGCB | GCB too small (AMGSBR). |
| 534 | G$ETIO | Workstation does not support TCB I/O redirection. |
| 535 | G$EDFL | Not enough disk space for raster device temporary file. |
| 900 | G$EANP | AMIGOS not present on system. |

# APPENDIX B

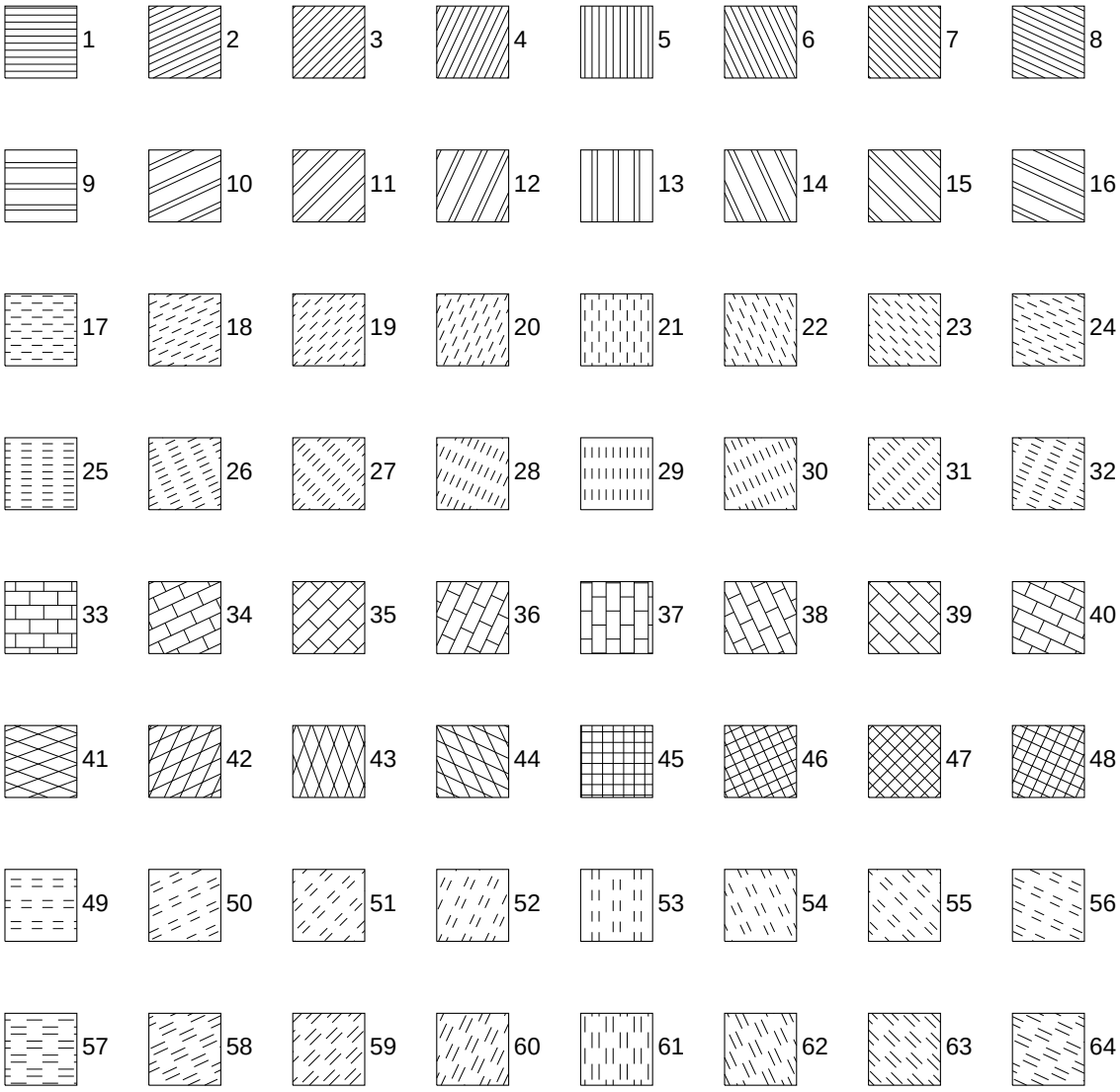
# DEFINED FILL AREA AND HATCH PATTERNS

The following fill area and hatch patterns have been pre-defined in AMIGOS. All additional workstation dependent patterns should be added at a starting value of 64 decimal since this list may expand in future releases of AMIGOS.

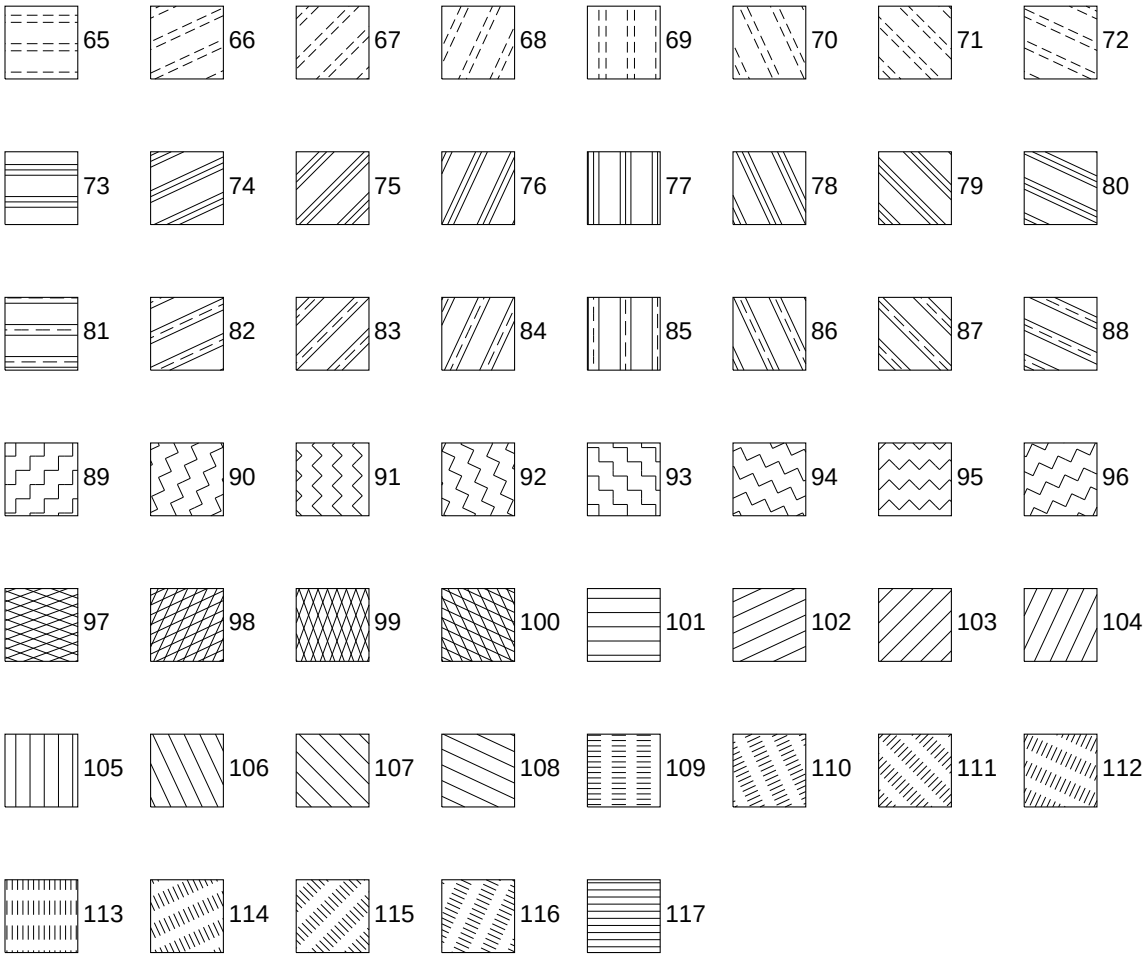
## FILL AREA PATTERNS



# HATCH PATTERNS



# HATCH PATTERNS (Continued)



# APPENDIX C

# BMP BITMAP IMAGE FILE FORMAT

This appendix describes the BMP bitmap image file format. This file format is designed to describe bitmap images of different sizes and resolutions in a device-independent fashion. Any device with bitmap display capabilities should be capable of displaying these bitmap images without extensive processing.

The BMP format provides:

- Variable size bitmaps, up to 65535 by 65535 pixels.
- Different size pixels, including 1, 4, 8 and 24 bits per pixel resolution for gray scale and color.
- Built-in color palette allowing up to 256 colors to be defined and associated with an image, with each color having a full 24-bit RGB resolution, or a full HLS color specification.
- The option of packing the image to reduce storage and transmission time requirements.

## C.1 THE BMP FORMAT

Each BMP file is a sequential data file containing four parts:

- A standard AMOS file header
- A bitmap definition block describing the bitmap image
- An optional color palette definition
- The image itself, which may be packed or unpacked.

### C.1.1 The AMOS File Header

Each BMP file starts with a standard AMOS file header, as generated by a PHDR statement. This allows a version number to be associated with the bitmap image that can be displayed by the DIR directory utility.

The standard file header occupies the first 10 bytes of the file.

### C.1.2 The Bitmap Image Definition Block

The bitmap image definition block consists of 70 bytes, the first 10 of which describe the overall image characteristics.

The first word contains the width of the image, in pixels. The second word contains the height of the image in pixels.

The third word contains the number of bits each pixel in the image occupies. Values of 1, 4, 8, and 24 may be specified.

The fourth word contains a series of flags describing other image attributes:

| | |
|---|---|
| Bit 0 | 1 if a palette is present, 0 if not. |
| Bit 1 | 1 if palette is HLS encoded, 0 if RGB encoded. |

The other bits is this word are reserved for future expansion.

The fifth word describes the packing algorithm used to pack the bitmap image. The supported values are:

| | |
|---|---|
| 0 | Image is unpacked. |
| 1 | Image is packed with run-length encoding, described below. |

Other values in this field are reserved for future expansion.

The next 60 bytes contain a null-terminated ASCII string describing the image. It is intended that this field be used in cataloging and organizing stored images.

### C.1.3 The Color Palette Definition Block

If a color palette is present in a BMP file (as defined within the bitmap definition block flags), it will immediately follow the definition block. If no palette is present, the bitmap image itself will immediately follow the definition block.

The first word of the palette definition contains the total number of palette indices minus one. Following this will be the palette indices themselves. If the bitmap definition block defines the palette as using RGB encoding, each palette index will consist of three bytes: one byte each for the R, G, and B color values, making a 24-bit color definition. If the bitmap definition block defines the palette as using HLS encoding, each palette index will consist of 6 bytes: two bytes each for the Hue, Lightness, and Saturation specifications.

The color palette is intended to be loaded into the display device before the image is displayed. The bitmap image itself will consist of a series of pixels whose values correspond to one of the color palette indices.

The first color palette index is always black and the second is always white. The remaining indices should be sorted in order of the frequency of usage of the color index within the bitmap image, with the most frequently used color immediately following the white index.

Keep in mind that while we refer to this as a color palette, the palette definition may in fact define a series of gray shades for the display of a monochrome image.

If no color palette is present in a BMP file, the display device's currently active color palette will be used.

**The Bitmap Image**

The remainder of the file contains the bitmap image itself. If the image is not packed (packing type 0 in the fifth word of the definition block), the image consists of a series of scan lines, ordered in a top-to-bottom, left-to-right fashion, describing the image. Each scan line occupies (image_width / (8/bits_per_pixel)) bytes. Each scan line starts on a byte boundary. The image height parameter defines the number of scan lines included in the image.

If the bitmap image is defined as packed, each scan line consists of a scan line width followed by a variable number of bytes. The scan line width parameter is a word value defining the number of following bytes that make up the current scan line. The following bytes make up the packed scan line, according to the packing/unpacking algorithm described below. All packing is done on a single scan line basis. No packing is done across scan lines. Each scan line starts on a byte boundary. The image height parameter defines the number of scan lines included in the image.

### C.1.4 The Image Packing Algorithm

Images are packed using a simple run-length encoding scheme which trades off packing density for ease and efficiency of the packing and unpacking operation, since these may be done within relatively "dumb" peripheral devices.

All image packing is done on the basis of a single scan line. Because the packing can result in a different number of packed bytes for each scan line, a byte count must precede each scan line.

Packing is done by compressing repeated bytes into a count and a single byte, and preceding any series of unique bytes with a count and the unique bytes. This technique has good worst case behavior in that it adds at most one byte for every 128 input bytes.

A pseudo-code fragment to unpack this type of image might be:

```
  Loop until all bytes are read in for this line
  Read the next source byte into n
  If n is between 0 and 127 inclusive, copy the next n+1 bytes literally
   ElseIf n is between -127 and -1 inclusive, copy the next byte-n+1 times
   ElseIf n is 128, do nothing
EndLoop
```

To pack an image, it is best to encode a two-byte repeat run as a replicate run, except when preceded and followed by a literal run, in which case it is better to merge all three runs into one literal run. Always encode three byte runs as replicate runs.

If a run is longer than 128 bytes, simply encode the remainder of the run as one or more additional replicate runs.

# APPENDIX D

# SAMPLE PROGRAMS

The function of these programs is to draw a filled, outlined box in the lower left of the workstation display. An example is provided for Assembler, AlphaC, and AlphaBASIC. In all cases, the workstation is assumed to be the user's terminal.

## D.1 ASSEMBLER SAMPLE PROGRAM

```
;DRWBOX      Program to draw a filled box using AMIGOS

  SEARCH     SYS                  ; get system definitions
  SEARCH     SYSSYM
  SEARCH     AMGSYM               ; get AMIGOS definitions


  VMAJOR=1.                       ; define revision level
  VMINOR=0.
  VEDIT=100.

PAGE
;Define Impure Area

.OFINI

.OFDEF  GCB,GC.SIZ                ; the GCB
; storage for polyline point array
.OFDEF  POLY,<5*4>+2
.OFSIZ  IMPSIZ

DRWBOX:
  PHDR       -1,0,PH$REU!PH$REE
  GETIMP     IMPSIZ,A5            ; get some impure area
  GOPWK      GCB(A5)              ; open the work station
  GCLRW      GCB(A5)              ; clear the screen
  GSFAS      GCB(A5),#2           ; set solid fill style
  GSFAC      GCB(A5),#4           ; set red fill color
  LEA        A2,POLY(A5)          ; index arg block
; Fill in the point array
  MOVW       #5,(A2)+             ; set number of points
```

```
MOVW      #0,(A2)+              ; set lower left coordinates
MOVW      #0,(A2)+              ;
MOVW      #0,(A2)+              ; set upper left coordinates
MOVW      #8000.,(A2)+
MOVW      #8000.,(A2)+          ; set upper right coordinates
MOVW      #8000.,(A2)+
MOVW      #8000.,(A2)+          ; set lower right coordinates
MOVW      #0,(A2)+
MOVW      #0,(A2)+              ; set lower left coordinates
MOVW      #0,(A2)+
GFA       GCB(A5),POLY(A5)      ; fill in the box
GPL       GCB(A5),POLY(A5)      ; outline the box with polyline
GCLWK     GCB(A5)               ; close the workstation
EXIT                            ; return to AMOS command level
END
```

## D.2 ALPHABASIC SAMPLE PROGRAM

```
!DRWBOX   Program to draw a filled box using AMIGOS

++INCLUDE AMGSYM.BSI    ! Get AMIGOS definitions

MAP1 GCB,X,20000        ! Allocate a graphics control block
MAP1 ERROR'STRING,S,30  ! Allocate a string for error message

! Define a polyline point array
MAP1 POINT'ARRAY
 MAP2 POINT'COUNT,B,2
 MAP2 POINTS(5)
  MAP3 POINT'X,B,2
  MAP3 POINT'Y,B,2

DRWBOX:

! Open the workstation defaulting to our terminal as output device
   XCALL AMGSBR,G'OPWK,GCB,"",STATUS
   IF STATUS <> 0 THEN GOTO AMGERR

! Clear the workstation
   XCALL AMGSBR,G'CLRW,GCB,STATUS
   IF STATUS <> 0 THEN GOTO AMGERR

! Set the fill area style to solid
   XCALL AMGSBR,G'SFAS,GCB,2,STATUS
   IF STATUS <> 0 THEN GOTO AMGERR

! Set the fill area color to red
   XCALL AMGSBR,G'SFAC,GCB,4,STATUS
   IF STATUS <> 0 THEN GOTO AMGERR

! Fill in the point array with desired coordinate points

   POINT'COUNT = 5
   POINT'X(1) = 0
   POINT'Y(1) = 0
   POINT'X(2) = 0
   POINT'Y(2) = 8000
   POINT'X(3) = 8000
   POINT'Y(3) = 8000
   POINT'X(4) = 8000
   POINT'Y(4) = 0
   POINT'X(5) = 0
   POINT'Y(5) = 0
```

```
! Fill in the box area
  XCALL AMGSBR,G'FA,GCB,POINT'ARRAY,STATUS
  IF STATUS <> 0 THEN GOTO AMGERR

! Now outline the box using the polyline function
  XCALL AMGSBR,G'PL,GCB,POINT'ARRAY,STATUS
  IF STATUS <> 0 THEN GOTO AMGERR

! Close the workstation
  XCALL AMGSBR,G'CLWK,GCB,STATUS
  IF STATUS <> 0 THEN GOTO AMGERR
  END

! process AMIGOS error
AMGERR:
  PRINT "AMIGOS error" ; STATUS ; "  " ;
! Get the error from AMIGOS
  XCALL AMGSBR,G'QERR,GCB,ERROR'STRING
  PRINT ERROR'STRING
  END
```

## D.3∞ALPHAC SAMPLE PROGRAM

```
/* Sample program to draw a box using AMIGOS
   Compile the program and link with AMGCLB.LIB */

#include <stdio.h>
#include <cdefs.c>
#include "amigos.h"

/* version "1.0(100),-1,0,PH$REE!PH$REU";   */

static struct gcb gcb;
static char *errpnt;
static struct
   {
     gword pcount;
     struct gpoint points[20];
   } parray;

main()
{
/*   printf ("gcb address is %o\n", &gcb);*/
     gcb.gc_flg = gcb.gc_flg|G_FERC|G_FBYP;      /* set return on error */
     if (gopwk(&gcb)) error_proc();              /* open workstation */
     if (gclrw(&gcb)) error_proc();              /* clear the workstation */
     if (gsfas(&gcb,2)) error_proc();            /* set solid fill */
     if (gsfac(&gcb,4)) error_proc();            /* set desired fill color */
     parray.pcount = 5;                          /* set 5 endpoints in array */
     parray.points[0].xcoord = 0;                /* Fill in the coordinates */
     parray.points[0].ycoord = 0;
     parray.points[1].xcoord = 0;
     parray.points[1].ycoord = 8000;
     parray.points[2].xcoord = 8000;
     parray.points[2].ycoord = 8000;
     parray.points[3].xcoord = 8000;
     parray.points[3].ycoord = 0;
     parray.points[4].xcoord = 0;
     parray.points[4].ycoord = 0;
     if (gfa(&gcb,&parray)) error_proc();        /* fill the box */
     if (gpl(&gcb,&parray)) error_proc();        /* outline the box */
     if (gclwk(&gcb)) error_proc();              /* close the workstation */
};

error_proc()
{
     printf("AMIGOS error %d.\n",gcb.gc_err);
     gqerr(&gcb,&errpnt);
     puts(errpnt);
     exit();
};
```

# GLOSSARY

**aspect ratio** - The width-to-height ratio of a rectangular area.

**attribute** - A property of a primitive determining its appearance, such as color.

**bitmap** - A generalization of an array of pixels on a raster device, it defines an array of rectangular cells with individual colors.

**character height** - The vertical extent of a character.

**clipping** - The process where portions of an image falling outside a window are discarded.

**color table** - A table associating color indices with actual colors shown on a display device.

**device coordinate** - A coordinate system specific to a display device. Most devices have a unique system of specifying positions on the display surface.

**device independence** - A characteristic of a computer software package enabling it to access and use different makes and models of display devices generating similar output.

**dynamic impure area** - An intermediate workspace used by AMIGOS which varies in size dependent on the graphic device driver in use.

**fill area** - A region in space with a well-defined boundary and whose interior may be distinguished in different ways.

**Generalized Drawing Primitive (GDP)** - GDPs are functions provided to access commonly used primitives which are not a part of the basic primitives, i.e., circles and ellipses.

**graphics control block (gcb)** - A memory work area used to store intermediate data specific to a particular workstation.

**Graphics Device Driver (GDV)** - A software module which translates commands and normalized device coordinates to those required by a specific device.

**linetype** - Used to distinguish different styles of lines, such as solid, dashed, and dash-dotted.

**linewidth** - A scale factor used with lines to determine the thickness of the line to be drawn.

**marker type** - Marker types are used to distinguish different marker symbols, such as dots, asterisks, and crosses.

**pixel** - The smallest element of a display surface that may be addressed independently. Literally, a pixel is a tiny dot.

**polyline** - A set of points joined by straight lines.

**Polymarker** - A set of positions marked by the same marker.

**primitive** - A basic unit of graphics output. A picture may be visualized to be made up of primitives, such as lines and markers.

**raster** - An array of pixels arranged in continuous rows and columns

**text** - Text in graphics refers to a string of characters displayed starting at a specified position.

**transformation** - The process of moving and scaling an image into a viewport.

**viewport** - A rectangular region specified in normalized device coordinates that determines the region into which a picture is projected.

**window** - A rectangular region specified in world coordinates in which you describe graphic objects.

**world coordinates** - A Cartesian coordinate system used by the application program to specify graphical data.

# DOCUMENT HISTORY

**Revision 00 - AMIGOS Release 1.0 - (Printed 8/88)**

New document for AMIGOS version 1.0.

**Revision 01 - AMIGOS Release 1.1 - (Printed 4/90)**

Documents the new AlphaC interface, and corrects minor, typographical errors.

# INDEX