PRO/GIDIS Manual

Order No. AA-HJ45A-TK

November 1985

This document describes PRO/GIDIS, DIGITAL'S General Image Display Instruction Set, as implemented for the PRO/Tool Kit. It is a user guide and reference manual for programmers developing graphics applications for the Professional.

REQUIRED SOFTWARE: Professional Host Tool Kit V3.0
or PRO/Tool Kit V3.0

OPERATING SYSTEM: P/OS V3.0
or RT-11 V5.2

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts 01754-2571

First Printing, December 1983
Updated, April 1984
Revised, November 1985

CONTENTS

FIGURES

TABLES

PREFACE

Manual Objectives

PRO/GIDIS is one of the tools you can use to develop graphics applications for the Professsional. This manual is both a user's guide and a reference manual for PRO/GIDIS, the General Image Display Instruction Set. It explains how to use PRO/GIDIS and describes each instruction in detail. It provides information about device-independent text and graphics programming with PRO/GIDIS.

Intended Audience

You should read this manual if you are developing a graphics application for the Professional and need information about PRO/GIDIS.

This document is intended for programmers who have had experience with systems programming and graphics applications software. You should also have experience with either MACRO-11 or FORTRAN.

This document explains how to use PRO/GIDIS on both the P/OS and RT-11 operating systems. All chapters except 4 and 5 apply to both operating systems. If you are using P/OS, read Chapter 4; if you are using RT-11, read Chapter 5.

Structure of This Document

This document has six chapters and eight appendixes.

Chapter 1, *Introduction to PRO/GIDIS*, describes PRO/GIDIS and places it in the context of other graphic tools. It provides guidelines so that you can determine whether to use PRO/GIDIS or some other graphics software.

Chapter 2, *Understanding PRO/GIDIS*, provides a conceptual framework for PRO/GIDIS. It explains key terms and introduces GIDIS instructions. Together with Chapter 4 (for P/OS), or Chapter 5 (for RT-11), this chapter serves as a user's guide.

Chapter 3, *PRO/GIDIS Syntax*, describes the GIDIS instruction syntax, which is the same for P/OS and RT-11.

Chapter 4, *Using PRO/GIDIS with P/OS*, explains how to use

PRO/GIDIS with P/OS, the Professional Operating System. The chapter describes the GIDIS Call Interface (GIDCAL), the devices accessed by GIDCAL, and error handling.

Chapter 5, *Using PRO/GIDIS with RT-11*, describes how to use PRO/GIDIS with the RT-11 operating system. The chapter describes three interfaces (including GIDCAL) and error handling.

Chapter 6, *PRO/GIDIS Instructions*, lists each GIDIS instruction in alphabetical order for quick reference. Information includes: format, arguments, notes explaining how the instruction works, and examples.

Appendix A, *PRO/GIDIS Instruction Summaries*, lists each PRO/GIDIS instruction, its operation code (opcode), argument length, opcode word, and associated arguments. Instructions are grouped two ways: by opcode and in alphabetical order.

Appendix B, *DEC Multinational Character Set*, shows the code table for the Professional's alphabet 0, the DEC Multinational Character Set.

Appendix C, *Font File Format*, describes the font file format required by the LOAD_BY_NAME instruction.

Appendix D, *Managing Fonts*, describes how to tell the font server about your font files.

Appendix E, *Area Texture and Color on the Plotter*, describes how Plotter GIDIS processes instructions that affect area texture and color.

Appendix F, *Alternate Access to Video GIDIS*, explains the Queue I/O Request (QIO$) and Queue I/O Request and Wait (QIOW$) system directives for P/OS. This access method is documented for backward compatibility with earlier versions.

Appendix G, *Glossary*, defines key terms used in this manual.

Associated Documents - P/OS

- *CORE Graphics Library Manual*
- *P/OS System Reference Manual*
- *RMS-11 Macro Programmers Guide*

- *PRO/Document VDM Manual*
- Tool Kit Language Manuals

Associated Documents - RT-11

- *RT-11 Programmer's Reference Manual*

Conventions Used in This Document

Convention/Term	Meaning
[optional]	In a command line, square brackets indicate that the enclosed item is optional. In a file specification, square brackets are part of the required syntax.
UPPERCASE	Uppercase words and letters indicate that you should type the word or letter exactly as shown.
lowercase	Lowercase words and letters indicate that you should substitute a word or value of your own. Usually the lowercase word identifies the type of substitution required.
...	A horizontal ellipsis indicates that you can repeat the preceding item one or more times. For example: parameter [,parameter...]
. . .	A vertical ellipsis means that not all of the statements are shown.
red	Interactive input appears in red.
Tool Kit	This general term refers to the software you use to develop applications to run on a Professional computer.
Host Tool Kit	The Host Tool Kit is Tool Kit software that runs on a host computer, rather than on the Professional itself.
PRO/Tool Kit	The PRO/Tool Kit is the Tool Kit software that runs on the Professional computer.

CHAPTER 1

INTRODUCTION TO PRO/GIDIS

PRO/GIDIS, the *General Image Display Instruction Set*, is one of several tools used to develop graphics applications for the Professional 300 Series computer. It consists of a set of instructions that provide the lowest-level, virtual device interface to the Professional's graphics hardware.

1.1 USES OF PRO/GIDIS

PRO/GIDIS is aimed at applications creating *compass and ruler* graphics, those in which images can be described using geometrical entities such as lines, arcs, and shaded areas. You can also use PRO/GIDIS to display mixed text and graphics. Figure 1-1 shows typical PRO/GIDIS output, a graphical representation of some sample statistical data.

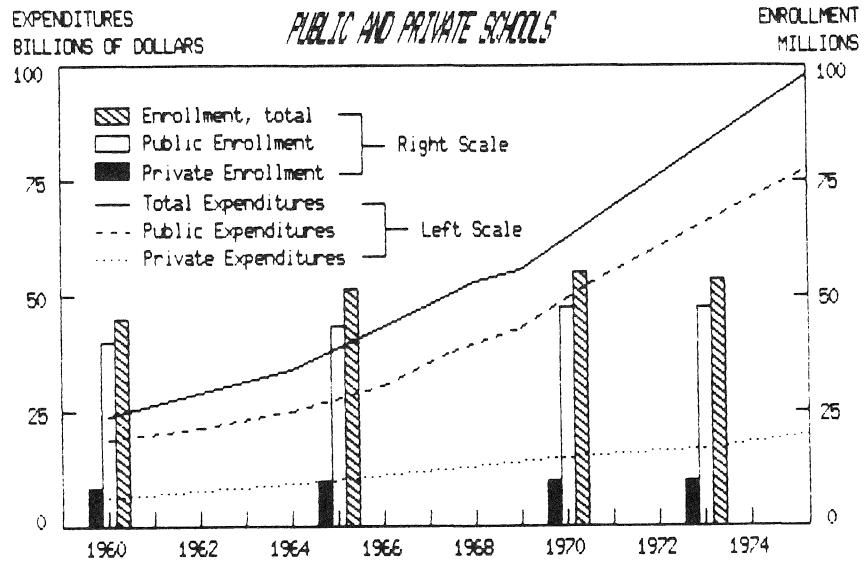


Figure 1-1: PRO/GIDIS Sample Output

PRO/GIDIS is the lowest layer of software that receives and interprets graphics instructions in a device-independent way. When the current output device cannot fully support an instruction, GIDIS provides an appropriate fallback.

With GIDIS under P/OS, you can write on a number of devices. Among them are the Professional video monitor, the LVP16 plotter, and various printers (the LN03, LA50 and LA100). You can also store GIDIS instructions in a file and later print the stored picture, either by itself or as part of a document. Under RT-11, you can write only on the video monitor.

The GIDIS Call Interface (GIDCAL) provides uniform access to each device supported by GIDIS. It also simplifies access to GIDIS from high-level languages.

1.2 RELATIONSHIP TO OTHER P/OS GRAPHICS TOOLS

PRO/GIDIS provides the foundation for several other graphics tools on the Professional. Because these tools are implemented as layers above PRO/GIDIS, each tool sets GIDIS attributes and expects to be in full control of them. As a result, use of more than one graphics protocol within an application is not supported.

Other graphics tools include:

- The PRO/Tool Kit CORE Graphics Library (CGL), a library of high-level graphics subroutines based on the ACM SIGGRAPH CORE Standard.

- ReGIS (Remote Graphics Instruction Set), a DIGITAL-developed, ASCII-based protocol, is used to transmit graphics instructions from a host computer to a remote Professional, VT125, VT240 or GIGI graphics terminal. A ReGIS to GIDIS converter (RTOG) translates ReGIS data files to GIDIS files that can be displayed (or printed) on the Professional. ReGIS currently cannot be used by applications that reside on the Professional itself; it can only be used in terminal emulation mode.

- NAPLPS, North American Presentation Level Protocol Syntax, is an ASCII-based protocol developed for Videotex/Teletext. NAPLPS currently cannot be used by applications that reside on the Professional itself; it can only be used in terminal emulation mode.

- TEK 4014 is an industry-standard Tektronix-based software protocol adapted from storage tube technology. TEK 4014 is available as a third party application that runs on the Professional only in terminal emulation mode.

- PRO/Document VDM, not a graphics tool itself, is the layer of P/OS that enables you to integrate graphics into documents.

Figure 1-2 shows the relationship between PRO/GIDIS and other graphic tools.

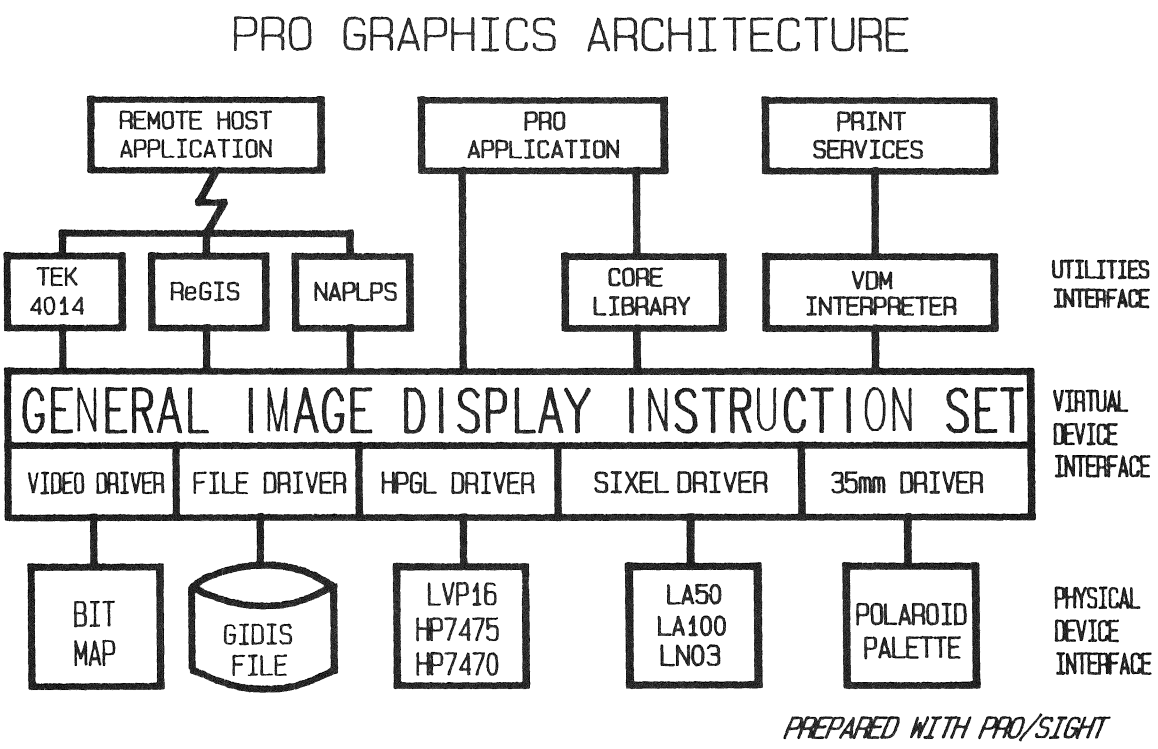


Figure 1-2: PRO/GIDIS Interface

1.2.1 When to Use PRO/GIDIS

Sometimes your choice of a graphics tool is a matter of taste, but there are some guidelines to go by.

- Use PRO/GIDIS if you want uniform access to the Professional's graphic devices.
- Use PRO/GIDIS if execution speed is most important.
- Use PRO/GIDIS to implement graphics utility layers, like CORE, or tools rather than applications.

1.2.2 When Not to Use PRO/GIDIS

Do not use PRO/GIDIS under the following conditions:

- If your program requires support for real (floating point) coordinates, curves, markers, and so forth, use the CORE Graphics Library.
- If you are concerned with portability of programs and industry-standard program interfaces to graphics routines, use the CORE Graphics Library.
- If you require VT100 or VT200 compatibility, use ReGIS with the Professional Terminal Emulator.

CHAPTER 2

UNDERSTANDING PRO/GIDIS

This chapter begins by briefly describing concepts in graphic programming. It then relates these concepts to GIDIS. Finally, it summarizes the types of instructions available in GIDIS.

2.1 INTRODUCTION TO GRAPHIC PROGRAMMING

Graphic systems typically provide the following functions:

- Viewing Transformation Instructions. These enable you to define your drawing area in coordinate units that are convenient for your application, then map the units to a device-independent coordinate system for displaying the image.

- Interactive Control Instructions. These enable you to interactively control how an image displays on a view surface. You can modify how a picture is mapped to a view surface, define cursors, scroll data, and output an image.

- Drawing Instructions. These enable you to draw figures within a picture.

- Attribute Instructions. These enable you to specify how the image appears when it displays.

The following sections describe the main functions and introduce terms commonly used in graphic programming.

2.1.1 Viewing Transformation Instructions

Two-dimensional graphic programming packages allow you to draw pictures in a Cartesian coordinate system, similar to drawing on graph paper. Most graphics systems allow you to define coordinate units that suit your particular application. Think of it as choosing graph paper with different scales, for example ten squares per inch versus fifteen squares per inch. These units are purely logical coordinates whose range is limited only by the arithmetic limits of the processor. Some systems allow floating point coordinates; others allow only integers. You draw pictures in user coordinate units that you define. All drawing instructions are stored in a database in user coordinates units. This user coordinate system is sometimes called the *World Coordinate System.*

Besides allowing you to create and store graphic data, a graphics system must have a way of displaying the contents of the database. (Display is used in a generic sense to include output to any device, not just screen display.) Because graphic output is displayed on a variety of output devices, a graphics system must have a way of mapping the user coordinates to a view surface. While a video monitor may be the most common view surface, printer and plotter output can also be considered a view surface.

The variety of output devices, both their shape and resolution, makes it desirable to have a device-independent way of describing the view surface. Hence, most graphics systems have a display coordinate system to describe the view surface. These coordinate system is sometimes called *Normalized Coordinate Space.* The exact way of defining the coordinate units within normalized space differs among graphic systems, but most allow you to choose coordinate units appropriate for any output device.

NOTE

To avoid confusion, this manual refers to operations performed in user coordinates as *drawing a picture.* It refers to operations performed in display coordinates as *displaying an image.*

Each graphic system performs the computations necessary to map the contents of the user coordinate system to the display coordinate system. This process of mapping from the user coordinate system to the display coordinate system is called the *viewing transformation.* However, the way the mapping proceeds differs from system to system. For example, when some graphic systems map the picture to the displayed image, distortion

results. Other systems preserve the shape of the picture. Some systems supply device drivers to complete the mapping in a way that preserves the image and suits the hardware requirements of the displaying device.

2.1.2 Interactive Control Instructions

Besides the standard viewing transformation operation, most systems provide interactive control instructions for modifying the mapping and manipulating the display.

Graphics systems differ in how much control they give you over the mapping process, for example controlling the size and shape of the displayed image. An explanation of how a graphics system gives you control over mapping requires the introduction of several more terms.

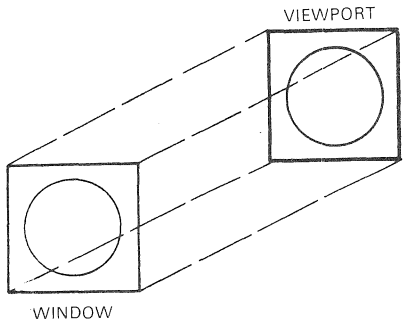
We refer to the entire contents of graphic data in the user coordinate space as a *picture.* You can map the entire picture to the view surface, or you can map only a portion of the picture to the view surface. You choose which portion to map by defining a *window*, a rectangular extent within the user coordinate space. By defining a window the same size as the picture, you map the entire picture to the view surface. By defining a window smaller than the picture, you map only a portion of the picture to the view surface. Only data within the defined window maps to the view surface. Anything outside the window is *clipped.* It remains in the picture, but is not displayed in the image on the view surface.

On the view surface, the image displays in a rectangular area called the *viewport.* The viewport is defined in display coordinates. Graphics systems allow you to define the viewport in a number of ways. For example, you can fill an entire view surface, providing you define your viewport as having the same shape as the output device. Or you can change the size and placement of a viewport. In some graphics systems, you can display more than one viewport simultaneously.

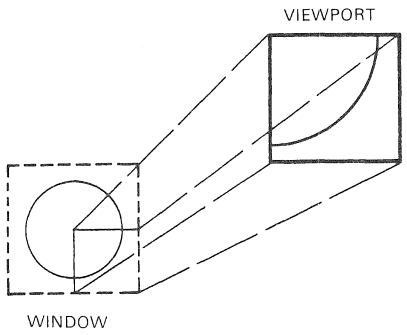
Because of all the options available both in defining the window and the viewport, mapping from user coordinates to display coordinates allows for many possibilities. You can, for example, define user and display coordinates to be identical and map an entire picture (the window encompasses the entire picture) to the entire viewport (which may or may not fill the display surface, depending on the shape of the viewport in relation to the shape of the display surface). Or you can define a window that includes only part of the picture, and map it to a larger viewport. This results in enlarging the image. Conversely, if

you define the window as larger than the picture and map it to a smaller viewport, you reduce the image. Besides affecting the size, enlarging or reducing the image also affects the granularity of the image.

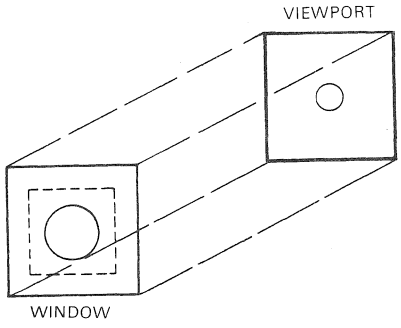
Figure 2-1 shows several mapping possibilities. Each case assumes a viewport that covers the entire view surface.



WINDOW, CLIPPING RECTANGLE, AND VIEWPORT SAME SIZE.
NO CLIPPING OF PICTURE.



WINDOW SMALLER THAN VIEWPORT, IMAGE ENLARGED, PICTURE CLIPPED.



WINDOW LARGER THAN VIEWPORT, IMAGE REDUCED, NO CLIPPING OF PICTURE.

MA-1148-85

Figure 2-1: Window to Viewport Mapping Options

Besides controlling mapping, graphics systems may also include instructions for using cursors or rubber bands to mark the current location. Other interactive control instructions enable you to erase, scroll, display, or print an image.

2.1.3 Drawing Instructions

Graphics systems provide you with building blocks to create a picture. These building blocks are called *output primitives*. Most systems have instructions for drawing points, lines, arcs, circles and text. Some also have instructions for filling figures, both closed and open figures. You build pictures by selecting appropriate drawing instructions.

2.1.4 Attribute Instructions

Graphics systems have attribute instructions that enable you to control how graphic output appears. Some attributes, like foreground and background color, affect all graphic output. Such attributes are called *global attributes*. Others affect only certain types of instructions, for example drawing lines or drawing text. These are typically called *line attributes* and *text attributes*, respectively. In most graphics systems, attributes are *modal*, that is they remain in effect until you explicitly change them.

2.2 INTRODUCTION TO GIDIS INSTRUCTIONS

GIDIS has the types of instructions common to most graphics systems, plus additional instructions for fonts and reports. This chapter describes GIDIS instructions in the following functional groupings:

- Picture Management Instructions. These provide the framework for creating and storing pictures, and for mapping them to an output device.

- Interactive Control Instructions. These include instructions for modifying the mapping process and manipulating the display.

- Drawing Instructions. These enable you to draw figures.

- Attribute Instructions. These enable you to specify how the figure appears when it displays.
- Alphabet and Font Instructions. These allow you to create alphabets and fonts.
- Report Instructions. These enable you to check the state of GIDIS.

2.2.1 Picture Management Instructions

Picture Management instructions provide a framework for defining pictures and set up the viewing transformation.

Because GIDIS attributes remain in effect until changed, you must include your specifications for the viewing transformation and all attributes in any picture you draw. The recommended way to do this is to frame all instructions for a given picture between a BEGIN_PICTURE and an END_PICTURE instruction.

In general, you use the Picture Management instructions as follows:

1. Use BEGIN_PICTURE to initiate definition of a picture.
2. You can use an INITIALIZE -1 next. This initializes GIDIS to its default values (see INITIALIZE in Chapter 6). Although it is more work, it is better practice to explicitly initialize each GIDIS attribute to a value of your own choice.
3. Set up an appropriate address space with SET_OUTPUT_IDS. Define coordinate values that are convenient for your application.
4. To control the appearance of your output, you should also set up the color map with SET_COLOR_MAP_ENTRY.
5. At this point you can use GIDIS attribute instructions and drawing instructions in any order you choose.
6. When you have finished, terminate the picture definition with an END_PICTURE.

The following paragraphs describe the GIDIS user and display coordinate spaces and how pictures are mapped to a view surface.

In GIDIS the user coordinate space is called *GIDIS Output Space (GOS)*. GOS units are limited to integers. The origin of GOS is the upper left-hand corner of the coordinate space. The pixel aspect ratio of X coordinate units to Y coordinate units is 1:1. All GIDIS instructions except SET_OUTPUT_IDS and SET_OUTPUT_VIEWPORT refer to GOS coordinates. You draw pictures and store them in GOS units. However, unless you use the Interactive Control instructions to alter the mapping process, you do not directly define a window in GOS coordinates. SET_OUTPUT_IDS defines the window and controls the mapping.

In GIDIS the display coordinate space is called *Imposed Device Space (IDS)*. Like GOS, the units are limited to integers, the origin is the upper left-hand corner of the coordinate space, and the pixel aspect ratio is 1:1. The left edge of the display surface is called the Y axis, and the top edge of the surface is called the X axis. You determine the extent of IDS by the coordinates you choose for the lower right-hand corner. You assign values to the bottom right-hand corner of the view surface with SET_OUTPUT_IDS.

You must always use a SET_OUTPUT_IDS instruction to set up a device- independent address space for displaying your image. SET_OUTPUT_IDS implicitly performs several other functions.

- It sets GIDIS Output Space (GOS) such that IDS and GOS units are identical. This means that the picture in GOS maps to the image in IDS identically.

- It sets your viewport to the entire view surface as defined by IDS. Your viewport is the rectangle (defined in IDS) within which the image is displayed on the view surface.

- It sets the clipping rectangle to the entire view surface as defined by IDS. The clipping rectangle is the window (defined in GOS) that contains the picture you want to map to the viewport. Thus, the window and viewport are identical.

The ability to define IDS in any coordinate units you choose allows you to control how your image displays in a device-independent way. Each output device has a certain shape (picture aspect ratio), resolution (number of physical pixels horizontally and vertically), and pixel aspect ratio (shape of physical pixel). These are hardware dependent. We call this hardware-dependent view *Hardware Address Space (HAS)*. For example, the Professional 350 video has a shape of 8 x 5 inches, a resolution of 960 horizontal by 240 vertical hardware pixels, and a pixel aspect ratio of 1:2.5. Because each X unit is not equal to each Y unit, the HAS is anisotropic. This means that you cannot map a coordinate system using a 1:1 ratio to the

Professional video without performing calculations to compensate for the distortion that would otherwise occur. The driver supplied for each of the supported output devices performs these adjustments.

You can choose, if you like, to tailor IDS for a particular output device. For example, if you want your image to fill the view surface, assign coordinate values that reflect the shape of the view surface. For example, if the view surface were 8 units wide by 5 units high, you might set [X,Y] of the bottom right corner to [79,49] or [799,499] or [959,599]. All these coordinates would fill the view surface and maintain the same shape. The only difference would be in the resolution. The more logical pixels (expressed in higher X and Y values), the finer the resolution of your drawing. In many cases, you will want to use the entire display surface.

If the shape you give IDS does not match the shape of the device's view surface, GIDIS starts at the upper left corner and maps as much as it can, leaving space on the bottom or to the right as necessary to maintain the proportions of your picture. This is why IDS is called device independent.

Figure 2-2 shows an example of a square IDS shape (with arbitrary coordinates of (500,500) that does not fill the view surface of an 8 by a 5-inch video display.

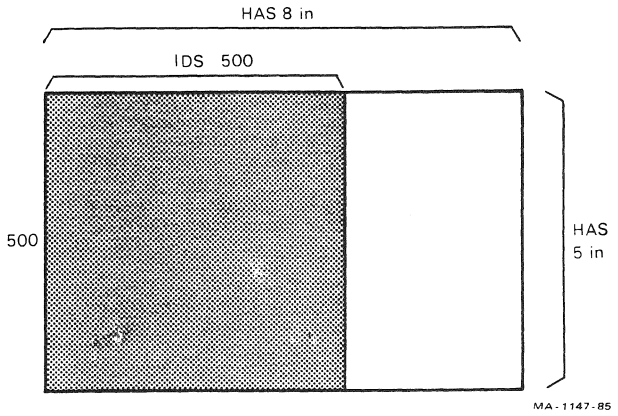


Figure 2-2: IDS Mapped onto a View Surface

You can use all picture management instructions either interactively or store them in a .GID file.

Table 2-1 lists the Picture Management instructions.

Table 2-1: Picture Management Instructions

Instruction	Action
NEW_PICTURE	Indicates the beginning of a new picture.
END_PICTURE	Indicates the end of a picture. Action depends on device.
INITIALIZE	Returns GIDIS to its power-up state. Aborts character, filled figure and picture definition blocks.
SET_OUTPUT_IDS	Specifies the coordinate units and shape of the image that displays on the view surface. Implicitly sets GOS, the clipping rectangle, and the viewport to be identical with IDS.
SET_COLOR_MAP_ENTRY	Sets red, green, blue mixture for the specified color map entry.

2.2.2 Interactive Control Instructions

These instructions control drawing operations within an interactive environment. Consequently, these instructions are inappropriate in a .GID file, a stored picture. Most interactive environments presume a video display.

Interactive applications should allow you to modify the display quickly and easily. GIDIS has interactive control instructions for modifying how an existing picture displays on the view surface and for drawing new pictures. When drawing new pictures, you need to be able to mark the current position, erase, scroll, output the picture to a view surface, and make a hard copy of the displayed image.

Several instructions control how an existing picture maps to a view surface. GIDIS allows you to display only part of a picture or change the size and location of your viewport.

If you want to display only a part of picture, use SET_GIDIS_OUTPUT_SPACE to define a coordinate extent smaller than the picture. This is useful to blow up a portion of a picture. For complete details, see SET_GIDIS_OUTPUT_SPACE in Chapter 6.

If you want to draw on only part of the view surface, use SET_OUTPUT_VIEWPORT to specify the size and location of your viewport. You can also specify multiple viewports and map a separate picture into each. For details, see Chapter 6.

Normally, your clipping rectangle equals your viewport. (SET_OUTPUT_IDS, SET_GIDIS_OUTPUT_SPACE, and SET_OUTPUT_VIEWPORT all set the clipping rectangle to match your viewport.) However, you can use SET_OUTPUT_CLIPPING_REGION to make your clipping rectangle smaller than your viewport. You might do this if you want to display a picture (or part of a picture) within a rectangle smaller than your viewport.

If you want to clear a rectangle within your viewport, set the clipping rectangle to the desired size and use ERASE_CLIPPING_REGION.

When using Video GIDIS, you may want to scroll (vertically or horizontally) whatever has been drawn within your clipping rectangle. Use SCROLL_CLIPPING_REGION to do this. The cleared space reverts to the current secondary color. Data scrolled out may not be scrolled back in; it must be redrawn.

While drawing a new picture with Video GIDIS, you may want to mark the current position. GIDIS gives you the option of using a cursor or rubber band to mark the current position. See SET_OUTPUT_CURSOR and SET_OUTPUT_RUBBER_BAND in Chapter 6. You select whether the cursor or rubber band blinks or is continuous with SET_OUTPUT_CURSOR_RENDITION.

When you want your application to execute all pending drawing instructions and prompt a user for further input, use FLUSH_BUFFER.

With the Professional 380 video, you can work with several pictures at a time. SET_OUTPUT_BITMAP enables you to draw up to four pictures (two in high resolution mode) in separate pages of the video bitmap. You can quickly move among them.

While drawing, you may want to print all or some portion of the video bitmap. PRINT_SCREEN allows you to send a specified portion of the video bitmap to a sixel printer connected to the printer port.

Table 2-2 summarizes the GIDIS Interactive Control Instructions.

Table 2-2: Interactive Control Instructions

Instruction	Action
SET_GIDIS_OUTPUT_SPACE	Specifies the coordinate units and shape of a window you define in GOS. Sets the clipping rectangle to coincide with the window.
SET_OUTPUT_VIEWPORT	Specifies the size and location of your viewport.
SET_OUTPUT_CLIPPING_REGION	Specifies the rectangle on the view surface where GIDIS can draw.
ERASE_CLIPPING_REGION	Clears clipping rectangle.
SCROLL_CLIPPING_REGION	In Video GIDIS, scrolls data within clipping rectangle.
SET_OUTPUT_CURSOR	Specifies the type of cursor used to mark the current position.
SET_OUTPUT_RUBBER_BAND	Specifies the type of rubber band used to mark the current position.
SET_OUTPUT_CURSOR_RENDITION	Selects whether the cursor or rubber band blinks or is continuous.

Instruction	Action
FLUSH_BUFFER	Executes any pending GIDIS instructions.
SET_OUTPUT_BITMAP	Selects bitmap on which to draw or display. (Professional 380 video only)
PRINT_SCREEN	Sends a specified portion of the video bitmap to a sixel printer connected to the printer port.

2.2.3 Drawing Instructions

GIDIS supplies the graphic primitives to draw lines, arcs, filled figures and text. You draw all pictures in GOS coordinates.

GIDIS drawing instructions can specify coordinates in either absolute or relative terms. Absolute terms are simply the X and Y coordinates you designate. Relative terms are in relation to the current position.

2.2.4 The Current Position

All GIDIS drawing instructions begin at the current position and end by setting a new current position. When you do not want the next drawing instruction to start where the last drawing instruction finished, use SET_POSITION or SET_REL_POSITION to move the current position to any point within GIDIS Output Space.

2.2.5 Drawing Lines, Arcs, Filled Figures, Characters, Images

You can draw one or a series of lines. DRAW_LINES and DRAW_REL_LINES draw from the current position to the specified position. When you use either instruction in a series, each *endpoint becomes the current position for the next line.*

You can draw arcs in much the same way with DRAW_ARCS or DRAW_REL_ARCS. All drawing begins at the current position and continues around a center point that you specify. As with drawing lines, you can draw arcs in a series, with each endpoint becoming the current position for the next arc. You determine

the direction and length of the arc by the angle. See Chapter 6 for details.

To draw a filled figure, you issue a BEGIN_FILLED_FIGURE instruction. You then use the instructions for drawing lines and arcs to designate the vertices of the figure. GIDIS stores the coordinate pairs for the vertices in the filled figure table. The order of the coordinates determines how the drawing proceeds. When GIDIS receives an END_FILLED_FIGURE instruction, it draws the filled figure. See Chapter 6 for limitations on the filled figure table.

To draw characters you must first have selected the current alphabet with a SET_ALPHABET instruction. Section 2.2.11 describes how to do this. Once you have a current alphabet, you indicate which character you want to draw by an index. GIDIS has two instructions for drawing characters. You can use DRAW_CHARACTERS for any alphabet, whether a standard one or one you design. You can use DRAW_PACKED_CHARACTERS for ASCII strings or any alphabet with fewer than 256 characters. With either instruction you can draw several characters in succession. The rendition of the characters is governed by the Text Attributes, described in Section 2.2.10.

Table 2-3 summarizes the GIDIS Drawing Instructions.

Table 2-3: Drawing Instructions

Instruction	Action
SET_POSITION	Moves the current position to an absolute point you specify.
SET_REL_POSITION	Moves the current position to a point you specify relative to the current position.
DRAW_LINES	Draws a line from the current position to an absolute point you specify.
DRAW_REL_LINES	Draws a line from the current position to a point you specify relative to the current position.

Instruction	Action
DRAW_ARCS	Draws an arc from the current position around an absolute center point you specify.
DRAW_REL_ARCS	Draws an arc from the current position around a center point you specify relative to the current position.
BEGIN_FILLED_FIGURE	Begins definition of a filled figure.
END_FILLED_FIGURE	Completes definition of a filled figure and draws the figure.
DRAW_CHARACTERS	Draws the character you specify.
DRAW_PACKED_CHARACTERS	Draws two characters you specify in one word.

2.2.6 Drawing Attributes

Several classes of attributes affect how your drawing looks. Some, namely the Writing Attributes, affect everything you draw. (The GIDIS Writing Attributes can be called global attributes.) Others, for example Line, Filled Figure, and Text Attributes, affect only certain drawing instructions. See Table 2-4 for a summary of the Drawing Attributes instructions.

When you power-up GIDIS, there are default values for GIDIS attributes. These default values make it possible to use the virtual device immediately. Table 6-4 lists the default values for GIDIS attributes. You can restore these default values at any time by using an INITIALIZE instruction.

However, you can specify your own values for these attributes by using the instructions explained in the following sections.

2.2.7 Writing Attributes

A drawing instruction operates on a pattern of ON and OFF bits (1 and 0 respectively). When you draw a line or arc, GIDIS derives the pattern from the line texture you specify. When you fill a

figure, GIDIS derives the pattern from the area texture you specify. When you draw a character, GIDIS derives the pattern from the raster image of the character. For example, the character "L" would be a horizontal and vertical line of 1's on a field of 0's.

```
0000000000
0100000000
0100000000
0100000000
0100000000
0100000000
0100000000
0111111100
0000000000
```

When you specify a pattern, you also specify its size in GOS units. The size controls how many times each bit in the pattern is repeated. For example, each 0 and 1 in the sample "L" may be repeated several times, depending on the size specified. When the pattern is displayed on a view surface, each bit in the pattern may be applied to multiple hardware pixels.

The writing attributes control how each drawing instruction interprets the pattern. There are four writing attributes: writing mode, primary color, secondary color, and plane mask.

Writing mode controls the Boolean operation performed on each bit of the pattern. For example, the default writing mode, overlay, works as follows. For each 1 in the pattern, GIDIS sets the current pixel to the primary color. For each 0 in the pattern, GIDIS leaves the current pixel unchanged. Your choice of writing mode affects how the image displays. See SET_WRITING_MODE in Chapter 6 for a full description of the writing modes provided by GIDIS.

SET_PRIMARY_COLOR specifies the color map index to use for all 1's in the bit pattern.

SET_SECONDARY_COLOR specifies the color map index to use for all 0's in the bit pattern.

SET_PLANE_MASK determines which planes are enabled for writing. Usually, you enable writing to all planes. This instruction ANDs (Boolean) the current color index and the plane mask (a representation of the planes you select). For the effect of a plane mask that is not set to all planes, see SET_PLANE_MASK in Chapter 6.

2.2.8 Line and Curve Attributes

You can choose to draw lines and curves with a solid or patterned line. With SET_LINE_TEXTURE you select the bit pattern that determines the appearance of the lines you draw.

You can also select the thickness of your drawing line with SET_PIXEL_SIZE. SET_PIXEL_SIZE sets the size of the logical pixel used as a paintbrush in subsequent drawing. The pixel is always a rectangle orthagonal to the x and y axes. Because of this, diagonal lines appear thicker than horizontal and vertical lines, except on a stroke device.

Figure 2-3 shows different pixel sizes used to draw a line.

Figure 2-3: Various Logical Pixel Sizes

2.2.9 Filled Figure Attributes

GIDIS allows you to select the two-dimensional pattern to be used in filling polygons. The pattern you choose is called the area texture cell. With the SET_AREA_TEXTURE instruction, you can choose either a character from an alphabet, or the current line texture as your area texture cell. Whatever pattern you choose remains the current area texture cell until you change it with another SET_AREA_TEXTURE.

You can choose a character from any alphabet, for example the default DEC Multinational Character Set, or an alphabet you create. Note, there is a 16 by 16 bit size restriction for a character used as a texture cell. However, with SET_AREA_TEXTURE_SIZE, you can enlarge the character used in filling a figure. GIDIS does this by multiplying the pattern in the texture cell. You can also clip unwanted white space from a text cell with SET_AREA_CELL_SIZE.

If you want a solid fill, specify a solid line with SET_LINE_TEXTURE and choose the current line texture as your area texture cell.

2.2.10 Text Attributes

With GIDIS Text Attribute instructions you control the size, spacing, orientation, and rendition (such as bold or italics) of text.

The GIDIS text model is based on the notion of *character cell.* A character cell is a rectangular field of ON and OFF bits. ON bits form a character pattern; OFF bits form the background. The character cell that stores the bit patterns can be up to 64 bits high and 64 bits wide.

You determine how the character cell is displayed by specifying the unit cell size and display cell size. SET_CELL_UNIT_SIZE specifies the size of the character you want displayed. GIDIS can scale the stored character cell to create larger or smaller characters. Scaling up is restricted to multiples of the bit pattern in the character cell.

SET_CELL_DISPLAY_SIZE gives you a way of extending the background field if you want. Having a display cell larger than the unit cell is an easy way to create white space between characters. You must always set both unit and display cell size, even if they are identical.

Besides setting a display cell width larger than a unit cell width to create white space, you can control spacing between character cells by specifying how to update the current position after a character is displayed. You have three choices:

- Implicit movement only. Specify implicit movement with SET_CELL_MOVEMENT_MODE and set explicit movement to (0,0) with SET_CELL_EXPLICIT_MOVEMENT. This causes the current position to move a display cell width along the current angle of cell rotation. If the current angle is 0, normal left to right text results.

- Explicit movement only. Specify no implicit movement with SET_CELL_MOVEMENT_MODE and set explicit movement to whatever you want with SET_CELL_EXPLICIT_MOVEMENT. For example, if you want upright characters drawn diagonally up to the right, set explicit movement to (n,-n). Note, however, that unless your explicit movement is greater than the display cell size, your characters overwrite each other.

- Implicit and explicit movement. Specify implicit movement with SET_CELL_MOVEMENT_MODE and explicit movement with SET_CELL_EXPLICIT_MOVEMENT. If you use both implicit and explicit movement, your characters move a display cell width plus whatever explicit movement you specify.

Figure 2-4 shows the three possibilities.

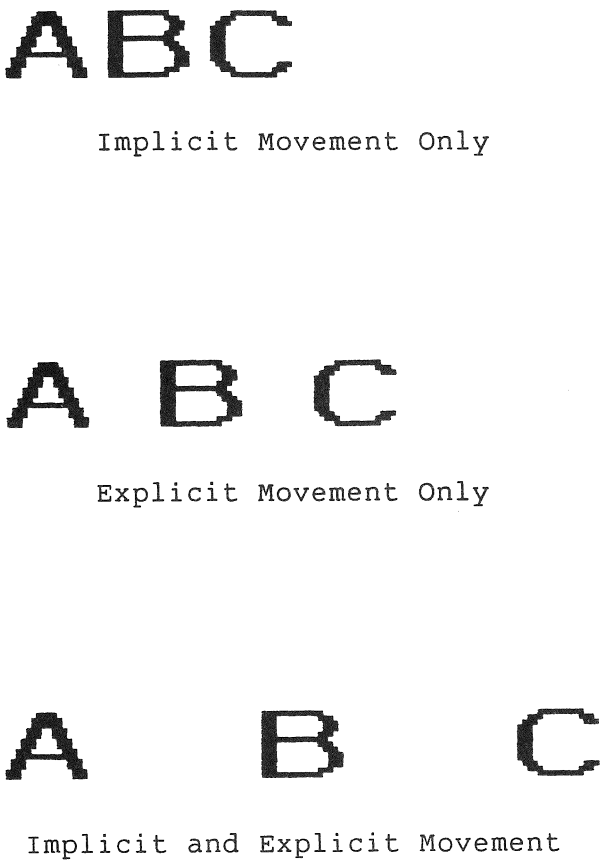


Figure 2-4: Implicit and Explicit Movement

GIDIS allows you to control how accurately the current position is updated. For device-independence and complete accuracy at the level of GOS, specify global symmetry. For best performance and constant intercharacter spacing, specify local symmetry. You specify symmetry with SET_CELL_MOVEMENT_MODE.

Accuracy and constant spacing are contradictory goals, because unit cell width may not be an integral number of hardware pixels. For example, suppose you specified a spacing of 25 GOS units, and the current output device had one hardware pixel for every two GOS units. With local symmetry, each character would move 24 GOS units. With global symmetry, each move would be 25 GOS units conceptually, but actually 12 pixels, then 13, 12, 13 and so on.

A character's orientation (the direction the character faces) depends on the angle of rotation as specified by SET_CELL_ROTATION. A character's angle of rotation is with respect to its top left corner. A positive angle rotates the left edge of the cell counter-clockwise; a negative angle rotates the left edge of the cell clockwise. The entire character cell rotates, without changing the shape of the cell. Figure 2-5 shows character cell rotation.

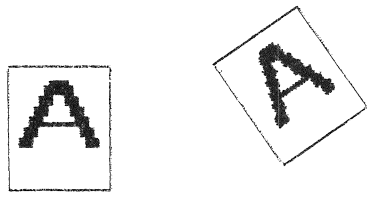


Figure 2-5: Character Cell Rotation

You can change the shape of the cell by using SET_CELL_OBLIQUE. When you specify a nonzero angle, the character cell becomes a parallelogram. A positive angle results in a back-slanted character; a negative angle in a front-slanted character.

You select various cell renditions by setting the appropriate flag with the SET_CELL_RENDITION instruction. If possible, GIDIS selects a font with the specified rendition. Otherwise, GIDIS algorithmically creates the specified rendition. You can specify the following rendition attributes: back-slant, italics, bold and proportional text.

Table 2-4 summarizes the GIDIS Drawing Attributes.

Table 2-4: GIDIS Drawing Attributes

Instruction	Action
Writing Attributes	
SET_PRIMARY_COLOR	Identifies the color map entry to use when drawing subsequent ON bits.
SET_SECONDARY_COLOR	Identifies the color map entry to use when drawing subsequent OFF bits.
SET_PLANE_MASK	Specifies which planes are accessible.
SET_WRITING_MODE	Selects writing mode to use in subsequent drawing.
Line and Curve Attributes	
SET_LINE_TEXTURE	Specifies the pattern used in drawing lines.
SET_PIXEL_SIZE	Specifies the thickness of the drawing line.

Instruction	Action
Filled Figure Attributes	
SET_AREA_TEXTURE	Selects the character to use as the texture cell in filling subsequent figures.
SET_AREA_TEXTURE_SIZE	Specifies the size to draw subsequent texture cells.
SET_AREA_CELL_SIZE	Clips or pads the last selected texture cell.
Text Attributes	
SET_CELL_UNIT_SIZE	Specifies the size to draw subsequent character cells.
SET_CELL_DISPLAY_SIZE	Specifies the size of a character's background field.
SET_CELL_EXPLICIT_MOVEMENT	Specifies the distance to move the current position after a character is drawn.
SET_CELL_MOVEMENT_MODE	Specifies how the current position moves after a character is drawn, and how accurately the current position is updated.
SET_CELL_OBLIQUE	Specifies how much to slant the character display cell.
SET_CELL_ROTATION	Defines the angle of rotation at which subsequent characters are drawn.
SET_CELL_RENDITION	Selects character renditions such as backslant, italics, bold, and proportional spacing.

2.2.11 Alphabets and Fonts

GIDIS uses the current alphabet in all text operations. To select an alphabet, use SET_ALPHABET. The selected alphabet remains current until you do another SET_ALPHABET.

A GIDIS alphabet is like an ASCII character set. When you specify an index within an alphabet, you know which particular character should be displayed. For example, the default alphabet (alphabet 0) is the DEC Multinational Character Set. When you specify index 101 (octal), you know that an uppercase "A" will be displayed.

A font, on the other hand, controls what the "A" looks like. A font's general appearance is denoted by typeface, for example Courier. A font has a rendition, for example roman, italic, bold, or bold italic. Fonts may be monospaced (each character cell has the same width) or proportionally spaced (character cell width varies with the character, for example the cell containing the character "m" is wider that the cell containing the character "i").

You create more than one font for an alphabet for improved quality. As Section 2.5 explained, GIDIS enables you to vary the appearance of text in a number of ways. GIDIS achieves these variations by either selecting a new font or algorithmically transforming the current font. Because there are limits to what can be effectively done by algorithmic transformation, you can ensure better quality by supplying a variety of fonts.

With GIDIS, you are not limited to standard alphabets and character sets. You can build your own alphabets and design your own *glyphs*, the graphic representations of each member of the alphabet. Section 2.2.13 explains how to do both. You may have up to 16 alphabets available at any time and an unlimited number of fonts. When you first select an alphabet from 1-15, it contains no characters. You fill the alphabet in one of two ways: you load a font file with LOAD_BY_NAME, or you dynamically create a font with CREATE_ALPHABET.

2.2.12 Font Files

A font file is simply a font that has been stored in a file. Appendix D explains how to name and store a font file so that the font server can use it.

You load a font file with the LOAD_BY_NAME instruction. LOAD_BY_NAME has two formats. Format 1 selects a specific font file. This format is primarily provided for compatibility with earlier versions of GIDIS. (See Chapter 6 for details.)

Format 2 (also called a family LOAD_BY_NAME) selects a typeface, known in GIDIS as a font family and identified by a family ID. When you do a Family LOAD_BY_NAME, you have really selected a pool of fonts. As you vary text attributes (such as unit cell size) and rendition attributes (such as bold), GIDIS automatically switches to the font file of the current family that best satisfies the attributes you have selected. (See Chapter 6 for details.)

2.2.13 Dynamically Created Fonts

You can build a font with CREATE_ALPHABET. This instruction establishes a storage cell size for each glyph in the font and the number of glyphs it contains. When you build a font with CREATE_ALPHABET you have two options for designing each glyph. With LOAD_CHARACTER_CELL you define a glyph by rows of bit patterns within a character cell. This method of defining glyphs is well-suited to raster devices.

With BEGIN_DEFINE_CHARACTER and END_DEFINE_CHARACTER, you create a glyph by drawing into the character cell with any of the GIDIS drawing instructions. All instructions between BEGIN_DEFINE_CHARACTER and END_DEFINE_CHARACTER draw into the character cell. This method of defining glyphs is well-suited to any device.

A font created dynamically with CREATE_ALPHABET has certain disadvantages.

- It takes time to build the font each time your application runs.
- The font remains defined only until you put another font into its alphabet.
- The font is stored in Read/Write memory. As a result, it is expensive to swap it to disk.

Thus, CREATE_ALPHABET should be used primarily for small, special alphabets like a set of patterns for filling figures.

If you are using P/OS, you can store a dynamically created font in a font file, by using the GIFONT routine of the GIDIS Call Interface. See Chapter 4 and Appendix D for details.

Table 2-5 summarizes GIDIS instructions for alphabets and fonts.

Table 2-5: Alphabet and Font Instructions

Instruction	Action
SET_ALPHABET	Selects the current alphabet.
LOAD_BY_NAME(1)	Loads the specified font file into the current alphabet.
LOAD_BY_NAME(2)	Associates the current alphabet with the specified font family.
CREATE_ALPHABET	Reserves storage space for a new alphabet font. Specifies the number of glyphs in the alphabet and the size of glyphs in the font.
LOAD_CHARACTER_CELL	Defines a glyph in terms of bit patterns within a character cell.
BEGIN_DEFINE_CHARACTER	Starts a character definition block. All subsequent instructions draw into the character cell.
END_DEFINE_CHARACTER	Completes a character definition block and draws the glyph.

2.2.14 Reports

You can ask GIDIS to generate various reports. You do this by issuing the appropriate request instruction. You read the report using GIREAD, as described in Chapter 4 (for P/OS) or Chapter 5 (for RT-11).

You can use reports to control program flow. For example, the position after a DRAW_ARCS or DRAW_CHARACTERS (local symmetry) may be different than what your program computes. You can check the actual current position with REQUEST_CURRENT_POSITION.

You can also use reports during debugging. In particular, every GIDIS instruction sets current status to SUCCESS or FAILURE. You may want to check current status after each GIDIS instruction when debugging. However, the cost of REQUEST_STATUS is too high for such use in a running application.

Table 2-6 summarizes all the GIDIS report generating instructions.

Table 2-6: Report Instructions

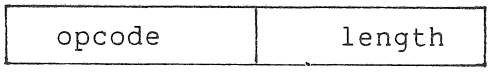
Instruction	Action
REQUEST_CELL_STANDARD	Reports in current GOS units the cell width and height to specify to generate standard size characters.
REQUEST_CURRENT_POSITION	Reports the current position.
REQUEST_OUTPUT_SIZE	Reports the attributes of the current device's view surface.
REQUEST_STATUS	Reports the success or failure of the last instruction.
REQUEST_VERSION_NUMBER	Reports characteristics of the current driver.

CHAPTER 3

PRO/GIDIS INSTRUCTION SYNTAX

The PRO/GIDIS interpreter accepts a stream of PRO/GIDIS instructions. An instruction consists of an operation code (opcode) word, and some number of argument words.

The format of an opcode word is:



Most GIDIS instructions require a fixed number of arguments. For example, SET_POSITION needs exactly two arguments.

Some GIDIS instructions accept a variable number of arguments, depending on whether optional arguments are included. Instructions in this category include: LOAD_BY_NAME and CREATE_ALPHABET. When an optional argument is omitted, GIDIS supplies a default as described in Chapter 6.

Some fixed length instructions are repeatable. You can repeat some of the arguments without repeating the opcode. For example, DRAW_REL_LINES X1, Y1, X2, Y2, X3, Y3 is equivalent to DRAW_REL_LINES X1, Y1 DRAW_REL_LINES X2, Y2, DRAW_REL_LINES X3, Y3. The instructions in this class include: DRAW_LINES, DRAW_REL_LINES, DRAW_ARCS, DRAW_REL_ARCS, DRAW_CHARACTERS, and DRAW_PACKED_CHARACTERS.

3.1 OPCODE BYTE

Each GIDIS instruction has a corresponding numeric code. For example, the INITIALIZE instruction has an opcode of 1, while the SET_PRIMARY_COLOR instruction opcode is 21. (Appendix A provides a list of PRO/GIDIS instructions and their corresponding opcodes.)

Your program can define PRO/GIDIS instruction names as numeric constants. For example, in MACRO-11, this could be:

```
G$INIT = 1.
G$PRIM = 21.
```

In FORTRAN, this could be:

```
INTEGER*2 GINIT,GPRIM
PARAMETER (GINIT = 1, GPRIM = 21)
```

In PASCAL, this could be:

```
CONST
  INITIALIZE = 1;
  SET_PRIMARY_COLOR = 21;
```

3.2 LENGTH BYTE AND THE ARGUMENT LIST

The length byte dictates the format of the instruction's argument list: counted or uncounted. Generally, you use a counted list for instructions with a fixed number of arguments, and an uncounted list for instructions with a variable number of arguments. However, you can use either a counted or uncounted argument list with any instruction.

A length value in the range 0 to 254 indicates a counted argument list. For example, if you specify a length value of two, PRO/GIDIS expects two argument words as shown below:

```
.BYTE   2.,29.  ;Instruction data block length = 2
                ;Opcode for SET_POSITION instruction = 29
.WORD   100.    ;x coordinate for current position
.WORD   350.    ;y coordinate for current position
                ;Following execution of this instruction,
                ;the current position is 100,350.
```

A length value of 255 indicates an uncounted argument list. Uncounted argument lists are terminated by an END_LIST instruction word (-32768), as shown below. Thus an argument word in an uncounted argument list cannot contain the value -32768.

```
.BYTE   255.,26.;introduces an uncounted argument list
                ;opcode for DRAW_REL_LINES
.WORD   10.     ;dx1
.WORD   -30.    ;dy1
.WORD   20.     ;dx2
.WORD   +60.    ;dy2
.WORD   -32768. ;END_LIST instruction opcode word
```

3.3 SYNTAX ERRORS

If GIDIS does not recognize an instruction opcode, it ignores that instruction and accompanying arguments. It also sets the status flag to FAILURE. If GIDIS encounters an instruction with insufficient arguments, it does not execute the instruction and sets the status flag to FAILURE. If GIDIS encounters an instruction with extra arguments, it executes the instruction as though the extra arguments did not exist.

For example, a SET_POSITION instruction with only one argument is ignored, while a SET_POSITION with three arguments is executed using only the first two arguments.

There are only two ways to confuse the GIDIS interpreter:

- Use END_LIST as an argument word in an uncounted argument list.
- Specify an argument count that differs from the actual number of arguments passed.

If you do either, you must reinitialize GIDIS. See the INITIALIZE instruction in Chapter 6.

CHAPTER 4

USING PRO/GIDIS WITH P/OS

This chapter describes how to use the GIDIS Call Interface (GIDCAL) with P/OS. It assumes you understand the conceptual framework of PRO/GIDIS (described in Chapter 2) and the PRO/GIDIS instruction syntax (described in Chapter 3).

- Section 3.1 describes the GIDIS Call Interface (GIDCAL).
- Section 3.2 describes the various devices accessed by GIDCAL.
- Section 3.3 explains how to build a task with GIDCAL.
- Section 3.4 documents GIDCAL error reporting.
- Section 3.5 lists sample programs for P/OS.

4.1 THE GIDIS CALL INTERFACE (GIDCAL)

The GIDIS call interface (GIDCAL) allows you to access each of the various GIDIS devices in the same way. GIDCAL consists of six routines:

- GIOPEN
- GIWRIT
- GIREAD
- GICLOS
- GIFONT

- GIPLAY

You access each routine by using the FORTRAN-compatible calling sequence (sometimes called the R5 calling convention). This means arguments are passed by reference, R5 is set to point to the argument list, and R1 through R5 are preserved by the called routine.

These standard routines make it easy for you to develop applications in high-level languages. You can use GIDCAL from MACRO-11 or any Tool Kit high-level language that supports FORTRAN-style calls.

Normally you use GIDCAL as follows:

1. Select the GIDIS driver you want to use with GIOPEN.

2. Pass GIDIS instructions with one or more calls to GIWRIT.

3. Read reports from REQUEST-type instructions, if any, with GIREAD.

4. Terminate the GIDIS connection with GICLOS.

Each GIDCAL routine returns a status code that indicates the results of the requested operation. If the operation is successful, a code of 1 is returned. If the operation is unsuccessful, a two-word error code block is returned. Section 4.4 explains how to interpret the codes.

You may have more than one GIDIS connection open at a time. This is useful if you want to print a GIDIS graphic while maintaining a connection to video GIDIS. GIDIS knows which driver to send instructions to by the Logical Unit Number (LUN) you specify with the GIOPEN call.

NOTE

Some high-level languages may reserve certain LUNs for their own use. If this is the case, you cannot access the same LUN. Check language documentation prior to assigning LUNs.

The following sections describe each GIDCAL routine and its arguments. The actual syntax for passing these arguments is specific to the high-level language you are using. See language documentation for details.

4.1.1 GIOPEN

GIOPEN initiates contact with the GIDIS driver of your choice. You choose a driver by specifying device type (Devtype) in the list of arguments. If you try to GIOPEN an active driver, Status is set to (-1,-7).

A GIOPEN does not affect the state of GIDIS. All attributes currently selected remain in force.

The list of arguments for GIOPEN follows.

GIOPEN (Status, LUN, Message, Msglen, Devtype, Driver)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
LUN	Unit-number associated with this GIOPEN. It should be an integer from 0 to 15. If not, Status is set to (-5,-1). If this LUN is already assigned to a GIDIS driver, Status is set to (-5,-4).
Message	Data to send to the driver when contact is initiated. Except as noted in Section 4.2, Message should be a word containing a 0.
Msglen	The number of words in Message. Except where noted, it should be 1. If Msglen is less than 0 or greater than 128, Status is set to (-5,-3).
Devtype	An integer that identifies the desired output device. If Devtype is invalid, Status is set to (-5,-2). If you try to GIOPEN a device for which there is no driver, Status is set to (-1,-2). The device types are: 0 - Disk File 1 - LA50 2 - LQP02 3 - LA100/LA210 4 - LVP16 5 - Other 6 - Video 7 - LN03 8 - Palette 9 - LQP03
Driver	Normally a 0. It should be nonzero only if you need to override the driver designated for the

device. (See Section 4.2 for driver names.) If you supply your own driver, identify it by the task name, in Radix-50.

Normally, the argument list for GIOPEN is (Status, LUN, 0, 1, Devtype, 0).

4.1.2 GIWRIT

GIWRIT outputs a buffer of GIDIS command data to the specified GIDIS driver. The data in a buffer does not have to start or end on a command boundary.

The list of arguments for GIWRIT follows.

GIWRIT (Status, LUN, Message, Msglen)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
LUN	Identifies the GIDIS driver to talk to. If no GIOPEN has been done for the specified value, Status is set to (-5,-1).
Message	The command data to send to the specified driver.
Msglen	The number of words in Message. If it is less than 0 or greater than 4095, Status is set to (-5,-3).

4.1.3 GIREAD

GIREAD waits for GIDIS to return the report and places it in the specified buffer. If the report is longer than the specified buffer, the end of the report is truncated. If the report is shorter than the specified buffer, the trailing words of the buffer are left unchanged.

The list of arguments for GIREAD follows.

GIREAD (Status, LUN, Buffer, Buflen)

Status	A two-word integer array used to return a code indicating the results of the requested operation.

LUN	Identifies the GIDIS driver sending the report. If no GIOPEN has been done for the specified device driver, Status is set to (-5,-1).
Buffer	Room for the report returned by GIDIS. Recall that the first word of a report contains a header specifying the type of report and the number of words in the buffer.
Buflen	The number of words in the report buffer.

4.1.4 GICLOS

GICLOS tells the specified GIDIS to end the connection. GICLOS does not return to its caller until the specified GIDIS has told it that all picture data has been output to the device.

A GIDIS driver processes a GICLOS by simulating an END_PICTURE instruction. (See Chapter 6 for details.) If the driver is not Video GIDIS, it exits when it has finished processing the picture.

If the driver is the type that buffers a picture before writing it, (for example, G$BITM) GICLOS causes picture output to commence if either:

- The user task **has not done** any END_PICTURE commands.
- The user task **has done** drawing commands since its last END_PICTURE.

The list of arguments for GICLOS follows.

GICLOS (Status, LUN)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
LUN	Identifies the GIDIS driver to terminate. If no GIOPEN has been done for the specified value, Status is set to (-5,-1).

4.1.5 GIFONT

GIFONT is independent of the other routines in GIDCAL. You use it to create a font file from the font loaded into alphabet 15. See CREATE_ALPHABET, SET_ALPHABET, BEGIN_DEFINE_CHARACTER, and LOAD_CHARACTER_CELL in Chapter 6 for information on how to create a GIDIS font.

The list of arguments for GIFONT follows.

GIFONT (Status, File spec, Len, Region name, Buffer, APR, LUN)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
File spec	Name of the font file you want created in ASCII. For example, "MYFONT.TSK."
Len	Number of characters in File spec.
Region name	Name (in Radix-50) to use for the font region when the font file is later used by GIDIS. See Appendixes C and D for details.
Buffer	256 word buffer that GIFONT uses as a temporary work area.
APR	APR that GIFONT maps alphabet fifteen's font into (8KB at a time).
LUN	Driver GIFONT should use when writing a font File spec

If you want to create a stroke font file (as opposed to a raster font file), you must run Plotter GIDIS. This ensures that the font is properly stored. To indicate a stroke font in the .FDF file (see Appendix D), specify a cell width and height of 1.

4.1.6 GIPLAY

Like GIFONT, GIPLAY is independent from the other GIDCAL routines. GIPLAY plays back the specified .GID file to the current output device, such as the video monitor. The file you want to play back must be on the local node. Only one task at a time can be doing a playback.

To use GIPLAY, you must first install the following file:

```
INSTALL [ZZSYS]CGLGRT.TSK
```

The list of arguments for GIPLAY follows:

GIPLAY (Status, LUN, File Spec, Len)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
LUN	Identifies the GIDIS driver writing the picture. If no GIOPEN has been done for the specified value, Status is set to (-5,-1).
File spec	Name of the file you want played back.
Len	Number of characters in File spec. A File spec can contain 1 to 59 characters. A length outside these bounds returns a Status of (-5,-5).

4.2 DEVICES ACCESSED BY GIDCAL

The Devtype value defined in a GIOPEN tells GIDIS which driver to access. Information about each device and its associated driver follows.

4.2.1 Disk File

The Message argument to GIOPEN should be the file specification that is the output device. There should be a null byte following the characters in the file spec. The Msglen argument to GIOPEN is the number of words in the file spec. Thus, whether the file specification is "A.GID" or "AB.GID", Msglen contains 3.

Calling GICLOS closes the file.

The driver is the task G$FILE.

4.2.2 LA50

The device area is assumed to be 8 inches wide by 10 and 2/3 inches high. The picture is automatically drawn to best fill the available area, so a landscape picture is drawn sideways.

Picture drawing starts when an END_PICTURE instruction is issued (or GICLOS simulates one).

The driver is the task G$BITM.

4.2.3 LQP02

No GIDIS driver is supplied for the LQP02. However, GIOPEN tries to access a task named G$LQP. If G$LQP does not exist, GIOPEN fails with Status set to (-1,-2).

4.2.4 LA100/LA210

The device area is assumed to be 8 inches wide by 10 and 2/3 inches high. The picture is automatically drawn to best fill the available area, so a landscape picture is drawn sideways.

Picture drawing starts when an END_PICTURE instruction is issued (or GICLOS simulates one).

The driver is the task G$BITM.

4.2.5 LVP16, HP7475, HP7470 Plotters

The user controls the device area by setting a dip switch. If set to A3, the area is about 17 inches wide by 11 inches high. If set to A4, the area is about 10 inches wide by 7.5 inches high. The picture is automatically drawn to best fill the available area, so a portrait picture is drawn sideways.

NOTE

The large paper size and portrait output *do not* apply to the HP7470.

The driver is the task G$HPGL.

4.2.6 Other Device

This device type is for accessing a private GIDIS. This allows third-party suppliers to develop alternative GIDIS devices. The format and content of the initialization message depend on the device supplier. However, we do suggest that suppliers allow a one-word message containing a zero.

No device driver is supplied for device type Other. However, GIOPEN tries to access a task named G$OTH. If the device supplier gives his GIDIS driver a different name than G$OTH, he must specify that name in the Driver argument to GIOPEN. Remember, the driver name should be the task name in Radix-50.

4.2.7 Professional Video

The device area is the entire screen. The screen is 8 units wide by 5 units high.

Picture drawing occurs as GIDIS instructions are received.

The driver is part of the Terminal Subsystem.

4.2.8 LN03

The device area is assumed to be 8 inches wide by 10 and 2/3 inches high. The picture is automatically drawn to best fill the available area, so a landscape picture is drawn sideways.

Picture drawing starts when an END_PICTURE instruction is issued (or GICLOS simulates one).

The driver is the task G$BITM.

4.2.9 Polaroid Palette

The device area is the entire print or slide. It is nominally 4 units wide by 3 units high.

Picture drawing starts when an END_PICTURE instruction is issued (or when GICLOS simulates one). During picture drawing, the Palette driver uses the video screen as a work area. When a slide camera is being used, GIOPEN opens the camera's shutter; GICLOS closes it and advances the film.

The driver is the task G$PAL. If it sets Status to (-6, any), it means a Palette I/O error has occurred. The second word of Status is the code returned by the Palette system. Table 4-1 lists the error codes, their meanings, and user actions.

Table 4-1: GIDCAL Palette Errors

Palette Code	Decimal Value	Error	User Action
"0"	48	Invalid Palette command	Report to Polaroid if the error recurs.
"1"	49	Invalid argument to Palette command	Report to Polaroid if the error recurs.
"2"	50	Filter wheel error	Report to Polaroid if the error recurs.
"3"	51	Communications error	Try readjusting RS-232 cable. If the error recurs, report to Polaroid.
"4"	52	No vertical sync	Try readjusting video cables and turning Palette off and on. If the error recurs, report to Polaroid.

4.2.10 LQP03

No GIDIS driver is supplied for the LQP03. However, GIOPEN tries to access a task named G$LQP. If G$LQP does not exist, GIOPEN fails with Status set to (-1,-2).

4.3 BUILDING A TASK WITH GIDCAL

GIDCAL is part of the PRO/Tool Kit. To link GIDCAL with your task, specify GIDCAL/LB just as you would for any other .OLB file. GIDCAL.OLB is on LB:[1,5]. GIDCAL, without GIFONT, uses about 800 words of your address space.

When you use GIFONT, you must include RMS in your task, plus room for the data area you pass to it.

Note the driver-specific instructions in Sections 4.4.1 and 4.4.2.

4.3.1 Video GIDIS

- When accessing Video GIDIS, GIDCAL uses one event flag (EFN). The default EFN is 29. If you want to give the EFN a different value, specify GBLDEF=GI$EFN:value in the task build command file.
- GIOPEN assigns the LUN you specified, if Devtype is Video.

4.3.2 Other GIDIS Drivers

- The GIDIS tasks G$BITM, G$HPGL, and G$PAL were built with an assigned ASG of the form ASG=LP:1. When one of these tasks starts up, it attaches the device associated with LUN 1; consequently, you cannot attach this device.
- Do not use the RSUM$ and SPND$ system directives with GIDCAL.
- GICLOS sends a one-word message that contains a -1. Therefore, you should not send such a buffer using GIWRIT.
- If you plan to access an LA50, LA100, or LN03, put INSTALL [ZZSYS]GIBITM in your installation file. If you plan to access Palette, put INSTALL [ZZSYS]GIPAL in your installation file. If you plan to access a private GIDIS, add the appropriate INSTALL command to your installation file.

4.4 ERROR REPORTING

All GIDCAL routines return a two-word status value. If the value of the first word is less than 0, an error was detected. The first word identifies the class of error; the second word identifies which error of the class has occurred. Table 4-2 lists the error classes and user actions to deal with the problem.

Table 4-2: GIDCAL Errors Listed by Class - P/OS

Code	Meaning	User Action
-1	Directive error	Refer to *RSX-11M/M-Plus Executive Reference Manual* for specific error.
-2	I/O Error	Refer to *IAS/RSX-11 Operations Reference Manual* for specific error.
-3	RMS Error	Refer to *RMS-11 Macro Programmer's Guide* for specific error.
-4	Internal error in GIDIS driver	Report error to DIGITAL.
-5	Interface error	Refer to Table 4-3 for specific errors.
-6	Palette driver error	Refer to Table 4-1 for specific error.

An error is driver-related if the first word of Status is anything but -5. This usually indicates a device problem (for example, the device is offline), but it could also mean that you passed a bad File spec to File GIDIS. Table 4-3 lists the types of errors for the value -5.

Table 4-3: GIDCAL Interface Errors - P/OS

Code	Error	User Action
-1	Invalid or unassigned LUN	Assign LUN with GIOPEN.
-2	Invalid device type	See Section 4.3.
-3	Improper message length	Assign Msglen within range.
-4	LUN already attached to a GIDIS driver	Select a new LUN.

4.5 SAMPLE P/OS PROGRAMS

4.5.1 Sample MACRO-11 Program

```
        .BLKW   2.
OBUF:   .BYTE   0.,55.  ;Length=0 REQUEST_CURRENT_POSITION
RBUF:   .BLKW   3.


        MOV #OARG, R5
        JSR PC,GIOPEN   ;SEND INSTRUCTION TO PRO/GIDIS
        TST     STAT
        BLE     ERROR   ;BRANCH IF GIOPEN FAILED

        MOV #WARG, R5
        JSR PC,GIWRIT   ;SEND INSTRUCTION TO PRO/GIDIS
        TST     STAT
        BLE     ERROR   ;BRANCH IF GIWRIT FAILED
                        ;READ THE REPORT
        MOV #RARG, R5
        JSR PC, GIREAD  ;READ THE REPORT
        TST     STAT
        BLE     ERROR   ;BRANCH IF GIREAD FAILED
                        ;
                        ; NEW CONTENTS OF RBUF:
                        ; BYTE AT RBUF    2.  (LENGTH)
                        ; BYTE AT RBUF+1  1.
                        ;   (CURRENT POSITION REPORT HDR)
                        ; RBUF+2:  CURRENT X POSITION
                        ; RBUF+4:  CURRENT Y POSITION

        MOV #CARG, R5
        JSR PC, GICLOS
        TST     STAT
        BLE     ERROR   ;BRANCH IF GICLOS FAILED

ERROR:                  ; Error handling routine

OARG:   .BYTE   6.,0
        .WORD   STAT
        .WORD   LUN
        .WORD   OPMSGL
        .WORD   OPMLEN
        .WORD   DEVTYP
        .WORD   DRIVER

WARG:   .BYTE   4.,0.
        .WORD   STAT
        .WORD   LUN
        .WORD   OBUF
        .WORD   MSGLEN
```

```
RARG:    .BYTE   4.,0.
         .WORD   STAT
         .WORD   LUN
         .WORD   RBUF
         .WORD   BUFLEN

CARG:    .BYTE   2.,0.
         .WORD   STAT
         .WORD   LUN

STAT:    .WORD   0.,0.
LUN:     .WORD   5.
OPMSGL:  .WORD   1.
OPMLEN:  .WORD   0.
DEVTYP:  .WORD   6.
DRIVER:  .WORD   0.
MSGLEN:  .WORD   1.
BUFLEN:  .WORD   3.
```

4.5.2 Sample FORTRAN Program

```
  INTEGER*2  OBUF
  INTEGER*2  RBUF(3),STAT(2)

  OBUF = 55*256+0    !OPCODE 55=REQUEST_CURRENT_POSITION
                     !LENGTH=0

  CALL GIWRIT (STAT, 5, OBUF, 1)
  IF (STAT.LE.0) GO TO 999       !BRANCH IF GIWRIT FAILED

  CALL GIREAD (STAT, 5, RBUF, 3)
  IF (STAT.LE.0) GO TO 999       !BRANCH IF GIREAD FAILED

          ! NEW CONTENTS OF RBUF:
          ! RBUF(1):  258 (i.e., 1*256+2 BECAUSE
          ! 1 = THE REPORT HDR AND 2 = LENGTH OF DATA FOLLOWING)
          ! RBUF(2):  CURRENT X POSITION IN GIDIS OUTPUT SPACE
          ! RBUF(3):  CURRENT Y POSITION IN GIDIS OUTPUT SPACE

999       ! ERROR FOUND
```

CHAPTER 5

USING PRO/GIDIS WITH RT-11

This chapter describes how to pass instructions to the GIDIS interpreter under RT-11. It assumes you understand the conceptual framework of PRO/GIDIS (described in Chapter 2) and the PRO/GIDIS instruction syntax (described in Chapter 3).

RT-11 requires that the FPU (floating point unit) hardware be installed on the Professional running PRO/GIDIS. RT-11 V5.2 runs PRO/GIDIS only as the foreground job under the XM monitor. Information in Chapter 6 about other devices does *not* apply to PRO/GIDIS under RT-11.

PRO/GIDIS requires two files: GIDIS.SAV and ALPH00.FNT. GIDIS.SAV is the utility save image. ALPH00.FNT is the default GIDIS font file. Both files must be on the system (SY:) device.

Issue the following command to start PRO/GIDIS and make it available to application programs:

```
.FRUN GIDIS.SAV
```

RT-11 provides software access to PRO/GIDIS using three interfaces.

- The GIDCAL interface (GIDIS call routines)
- The MACRO-11 interface (.SPFUN programmed request)
- The FORTRAN interface (ISPFN/ISPFNC/ISPFNF/ISPFNW)

The Professional Interface (PI) handler controls the operation of PRO/GIDIS and is transparent to the user. PRO/GIDIS instructions from application programs are sent to and received from PI using any of the above interfaces.

5.1 THE GIDIS CALL INTERFACE (GIDCAL)

Under RT-11, the GIDCAL routines consist of four FORTRAN system subroutines:

- GIOPEN
- GIWRIT
- GIREAD
- GICLOS

The subroutines are located in the system subroutine library SYSLIB.OBJ.

With the following exceptions, the GIDCAL routines work the same under RT-11 as they do under P/OS.

- GIFONT and GIPLAY, two GIDCAL routines available under P/OS, are not currently supported.
- For RT-11 V5.2, GIDCAL addresses only the PRO Video (Devtype 6).
- In GIWRIT the maximum message length (msglen) is 2048 decimal words.

Normally you would use GIDCAL as follows:

1. Initiate the GIDIS operation with GIOPEN.
2. Pass GIDIS instructions with one or more calls to GIWRIT.
3. Read reports from REQUEST-type instructions, if any, with GIREAD.
4. Terminate the GIDIS connection with GICLOS.

Each GIDCAL routine returns a status code that indicates the results of the requested operation. If the operation is successful, a code of 1 is returned. If the operation is unsuccessful, a two-word error code block is returned. Section 5.1.5 explains how to interpret the codes.

NOTE

Some high-level languages may reserve certain LUNs for their own use. If this is the case, you cannot access the same LUN. Check language documentation prior to assigning LUNs.

The following sections describe each GIDCAL routine and its arguments. The actual syntax for passing these arguments is specific to the high-level language you are using. See language documentation for details.

5.1.1 GIOPEN

GIOPEN initiates contact with the Professional interface (PI) handler and assigns a logical unit number (LUN) for this GIDIS operation. A GIOPEN does not affect the state of GIDIS. All attributes currently selected remain in force.

To initialize the Professional video screen, execute the INITIALIZE -1 (complete initialization) instruction, followed by the NEW_PICTURE instruction.

The list of arguments for GIOPEN follows.

GIOPEN (Status, LUN, Message, Msglen, Devtype, Driver)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
LUN	Unit-number associated with this GIOPEN. It should be an integer from 0 to 15. If not, Status is set to (-5,-1). If this LUN is already connected to a GIDIS operation, Status is set to (-5,-4).
Message	Data to send to Video GIDIS. Message should be a word containing a 0.
Msglen	The number of words in Message. Except where noted, it should be 1. If Msglen is less than 0 or greater than 128, Status is set to (-5,-3).
Devtype	An integer that identifies the desired output device. For RT-11 V5.2 only Devtype 6 is valid. Integer values 0 through 5, 7 and 8 are reserved. If Devtype is invalid, Status is set to (-5,-2).
Driver	0, as RT-11 accesses only video GIDIS

Normally, the argument list for GIOPEN is (Status, LUN, 0, 1, Devtype, 0).

5.1.2 GIWRIT

GIWRIT outputs a buffer of GIDIS command data to the specified GIDIS driver. The data in a buffer does not have to start or end on a command boundary. The list of arguments for GIWRIT follows.

GIWRIT (Status, LUN, Message, Msglen)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
LUN	Identifies the unit number assigned by GIOPEN. If no GIOPEN has been done for the specified value, Status is set to (-5,-1).
Message	The command data to send to Video GIDIS
Msglen	The number of words in Message. If it is less than 0 or greater than 2048 (decimal words), Status is set to (-5,-3).

5.1.3 GIREAD

GIREAD waits for GIDIS to return the report and places it in the specified buffer. If the report is longer than the specified buffer, the end of the report is truncated. If the report is shorter than the specified buffer, the trailing words of the buffer are left unchanged.

The list of arguments for GIREAD is as follows.

GIREAD (Status, LUN, Buffer, Buflen)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
LUN	The unit number assigned by GIOPEN. If no GIOPEN has been done for the specified device driver, Status is set to (-5,-1).
Buffer	Room for the report returned by GIDIS. Recall that the first word of a report contains a header specifying the type of report and the number of words in the buffer.
Buflen	The number of words in the report buffer.

5.1.4 GICLOS

GICLOS ends the GIDIS connection to the Professional interface handler. The output device treats a GICLOS subroutine as an END_PICTURE instruction. Control is returned to the calling program once all data specified by the GIWRIT subroutine has been sent to the output device. (See Chapter 6 for details.)

The list of arguments for GICLOS is as follows.

GICLOS (Status, LUN)

Status	A two-word integer array used to return a code indicating the results of the requested operation.
LUN	The unit number to terminate. If no GIOPEN has been done for the specified value, Status is set to (-5,-1).

5.1.5 GIDCAL Error Reporting

GIDCAL subroutines can return the following error codes and subcodes in the two-word status array. The first word specifies the class of the error; the second word specifies the type of error within the class.

GIDCA running under RT-11 returns three classes of errors listed in Table 5-1.

Table 5-1: GIDCAL Errors Listed by Class - RT-11

Code	Meaning
-1	Directive error
-5	Interface error
-7	Operating System Error

The directive error code (-1) can return the following subcode:

-1 No handler. The output device handler is not loaded.

The interface error code (-5) returns the subcodes listed in Table 5-2.

Table 5-2: GIDCAL Interface Errors - RT-11

Code	Error
-1	Invalid or unassigned LUN
-2	Invalid device type
-3	Improper message length
-4	LUN already attached to a GIDIS driver

In addition to the directive and interface errors, RT-11 also reports operating system errors (-7). Table 5-3 describes the specific errors within this class.

Table 5-3: RT-11 Operating System Errors

Code	Error
Codes returned during a GIDIS operation	
-1	Required argument missing. A required argument in a GIDCAL subroutine is not specified.
-2	Handler not found. The indicated file was not found on the device.
-3	File not found. The indicated file was not found on the device.

Code	Error
-4	File open on nonsharable or non-file-structured device.
-5	An attempt was made to read or write past the end-of-file (EOF) mark.
-6	Hard error. The GIDIS operation experienced a hard error on the output device.
Codes returned when the .SERR programmed request is in effect.	
-129	Called USR from completion routine.
-130	No device handler; this operation needs one.
-131	Error doing directory I/O.
-132	.FETCH error. An I/O error occurred while the handler was being used, or an attempt was made to load the handler over USR or RMON.
-133	Error reading an overlay.
-134	No more room for files in the directory.
-135	Reserved.
-136	Invalid channel number; number is greater than actual number of channels that exist.
-137	Invalid EMT, and invalid function code has been decoded.
-138	Reserved.
-139	Reserved.
-140	Invalid directory.
-141	Unloaded XM handler.
-142	Reserved.
-143	Reserved.
-144	Reserved.
-145	Reserved.

Code	Error
-146	Reserved.

5.1.6 Sample Program Using GIDIS Call Interface

The following FORTRAN program fragment uses the GIDCAL subroutines to request the current cursor position.

```
C
C       Declare storage.
C
        INTEGER*2       BUFLEN , LUN , MSGLEN , OCLEN , OPCODE
        INTEGER*2       BUFFER( 3 ) , MESSAG( 1 ) , STATUS( 2 )
C
C       User program begins here...
C
        .
        .
        .
C
C       Assign Logical Unit Number.
C
        LUN     =       5
C
C       Assign opcode (REQUEST_CURRENT_POSITION) and opcode
C       length (0).
C
        OPCODE  =       55*256
        OCLEN   =       0
C
C       Insert opcode and opcode length into message buffer
C       (one word).
C
        MESSAG( 1 )     = OPCODE + OCLEN
        MSGLEN          = 1
C
C       Send the message to GIDIS
C
        CALL    GIWRIT( STATUS , LUN , MESSAG , MSGLEN )
C
C       Check for errors.
C
        IF      ( STATUS( 1 ) .LE. 0 ) GOTO 999
C
C       Assign buffer length for report.
```

```
C
      BUFLEN  =       3
C
C     Get a report from GIDIS.
C
      CALL    GIREAD( STATUS , LUN , BUFFER , BUFLEN )
C
C     Check for errors.
C
      IF      ( STATUS( 1 ) .LE. 0 ) GOTO 999
C
C     Contents of BUFFER after successful return:
C
C     BUFFER( 1 ) = 258 ( (1*256) + 2 )
C                       1 = Report header,
C                       2 = Number of data elements in buffer
C     BUFFER( 2 ) = Current "X" position in GIDIS output space
C     BUFFER( 3 ) = Current "Y" position in GIDIS output space
C
C
C     User program continues from here...
C
      .
      .
      .
C
C     Handle errors.
C
  999 ...
C
C     End of GIDCAL example.
C
      END
```

5.2 THE MACRO-11 PRO/GIDIS INTERFACE

With the MACRO-11 interface, PRO/GIDIS instructions from application programs are sent to and received from the Professional interface handler using the .SPFUN programmed request. The .SPFUN programmed request is located in the distributed RT-11 MACRO library SYSMAC.SML.

RT-11 supports PRO/GIDIS from MACRO-11 or any supported high-level language that uses external MACRO-11 routines. The recommended method is to write callable MACRO-11 routines that issue the .SPFUN programmed request. For information on calling the .SPFUN programmed request from a supported high-level language, refer to the documentation for that language.

When programming for GIDIS using the .SPFUN request of ISPFN subroutines, you should initialize GIDIS before sending it your GIDIS instructions. Perform the following operations each time you begin a new program:

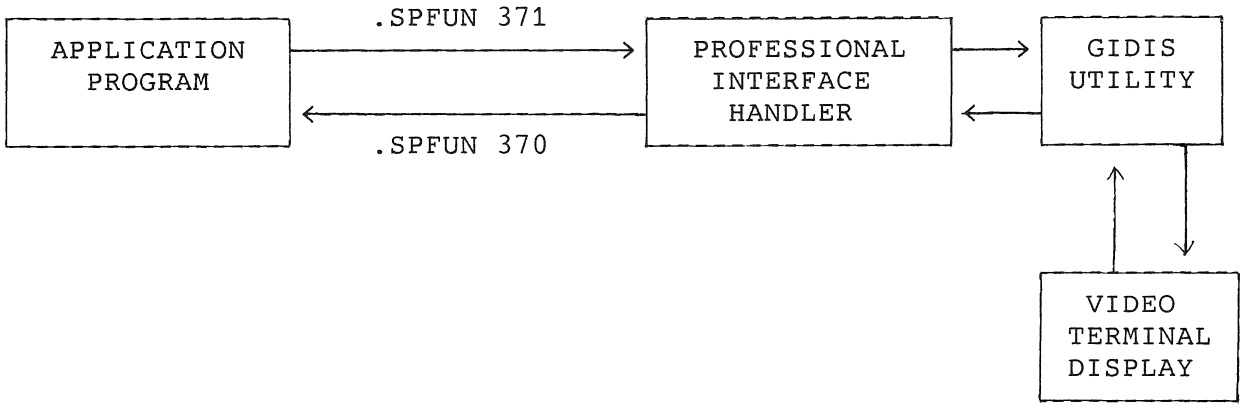
1. Establish a channel to PI with the .LOOKUP request.

2. Issue an .SPFUN 371 request and specify -1 for the wcnt argument.

3. Issue an .SPFUN 371 with the INITIALIZE instruction.

4. Issue the .SPFUN 371 that writes your data buffer to GIDIS.

RT-11 requires PRO/GIDIS to be the highest priority job.
The following is a simplified illustration of the RT-11 PRO/GIDIS data path:



5.2.1 .SPFUN 371

The .SPFUN 371 writes (sends) one or more PRO/GIDIS instructions and their associated parameter values to the Professional interface handler in a buffer. The buffer must begin at an even address. The Professional interface handler passes the buffer to the GIDIS utility for processing. You can pass a maximum of 2048 (decimal) words to the PI handler in one .SPFUN 371 request.

The following is the structure of the .SPFUN 371 programmed request when used with the Professional interface handler.

Macro Call: .SPFUN area,chan,func,buf,wcnt,blk

area — Is the address of a six-word EMT argument block

chan — Is the channel number in the range 0 to 376 (octal)

func — Is 371

buf — Is the address of the buffer containing the input to the GIDIS utility. Buf must start on a word boundary

bcnt — Is the number of bytes of information being sent

blk — Is zero

The .SPFUN 371 request can return error codes; see the *RT-11 Programmer's Reference Manual* for complete information.

Issuing a REQUEST_STATUS instruction returns a report on the success or failure of an instruction sent by .SPFUN 371. Check the carry bit on return from .SPFUN 371 to determine whether the instruction was successfully sent to PRO/GIDIS.

5.2.2 .SPFUN 370

The .SPFUN 370 reads (returns) a buffer of information generated from a PRO/GIDIS REQUEST-type instruction (sent using .SPFUN 371). The buffer must begin at an even address. The Professional interface handler passes the buffer address to PRO/GIDIS, and PRO/GIDIS loads the information into the buffer.

The following is the structure of the .SPFUN 370 programmed request when used with the Professional interface handler.

Macro Call:	.SPFUN area,chan,func,buf,bcnt,blk
area	Is the address of a six-word EMT argument block
chan	Is the channel number in the range 0 to 376 (octal)
func	Is 370
buf	Is the address of the buffer containing the input to the GIDIS utility. Buf must start on a word boundary
wcnt	Is the maximum number of words the GIDIS utility can place in the buffer
blk	Is zero

The .SPFUN 370 request can return error codes; see the *RT-11 Programmer's Reference Manual* for complete information.

5.2.3 SAMPLE MACRO-11 PROGRAM

The following example returns the current position of the cursor.

```
        G$RCP=:55.                      ; Specify instruction
        G$INT=: 1.                      ; codes

        .LOOKUP #IOAREA,#0,#PIBLK       ; Open PI on channel 0
        BCS     ERROR                   ; Check for success
                                        ;
        .SPFUN  #IOAREA,#0,#371,,#-1,#0
                                        ; Initialize GIDIS
        BCS     ERROR                   ; Check for success
                                        ;

        .SPFUN  #IOAREA,#0,#371,#REQPOS,#3,#0
                                        ; Send the instructions
                                        ; to initialize GIDIS
                                        ; internal symbols and
                                        ; REQUEST_CURRENT_POSITION.
                                        ;
        BCS     ERROR                   ; Check for success
                                        ;
        .SPFUN  #IOAREA,#0,#370,#REPBUF,#3,#0
                                        ; Read the current
                                        ; position.
        BCS     ERROR                   ; Check for success
                                        ;
```

```
                ; .SPFUN 370 causes the following report to be
                ; placed in REPBUF:
                ;
                ; BYTE 2.      (number of data words following).
                ; BYTE 1.      (CURRENT_POSITION_REPORT
                ;               identifier).
                ; WORD x       (PRO/GIDIS coordinates
                ; WORD y        for current position).
                ;
                ; The current position of the cursor will be in
                ; the second and third words of REPBUF.
        .
        .
        .
IOAREA: .BLKW   6         ; .SPFUN EMT argument block
PIBLK:  .RAD50  /PI /     ; File name in Radix-50 characters
        .WORD   0,0,0     ;
REQPOS: .BYTE   1,G$INT   ; Length=1, opcode = INITIALIZE
        .WORD   -1        ; Initialize operand
        .BYTE   0,G$RCP   ; Length=0,
                          ; opcode = REQUEST_CURRENT_POSITION.
REPBUF: .BLKB   6         ; Buffer for info returned from GIDIS.
ERROR:                    ; Error handling routine.
```

5.3 THE FORTRAN PRO/GIDIS INTERFACE

FORTRAN provides its own system subroutines (ISPFN/ISPFNC/ISPFNF/ISPFNW) that are used in the same manner as the MACRO-11 .SPFUN programmed requests. These subroutines are described in Chapter 3 of the *RT-11 Programmer's Reference Manual*. The four subroutines are located in the distributed RT-11 system subroutine library SYSLIB.OBJ.

Follow the order of operations described in Section 5.2.

A sample FORTRAN program using the ISPFNW system subroutine follows.

SAMPLE FORTRAN PROGRAM

The following example returns the current position of the cursor.

```
C
C       Sample FORTRAN program for PRO/GIDIS.
C
C       Declare storage.
C
        INTEGER*2       RDCPOS , RQCPOS
```

```
      INTEGER*2       BLOCK , CHAN , STATUS , WCNT
      INTEGER*2       FILSPC( 4 )

      BYTE            REPBUF( 6 ) , REQBUF( 2 )

      DATA            FILSPC/ 3RPI , 0 , 0 , 0 /
C
C     Assign SPFUN function codes ( Read, Request ).
C
      RDCPOS  =       "370
      RQCPOS  =       "371
C
C     Initialize default values.
C
      BLOCK   =       0
C
C     Get an RT-11 channel.
C
      STATUS  =       IGETC()
      IF              ( STATUS .EQ. -1 ) GOTO 900
      CHAN    =       STATUS
C
C     Open the PI handler.
C
      STATUS  =       LOOKUP( CHAN , FILSPC )
      IF              ( STATUS .NE. 0 ) GOTO 910
C
C     Send the instruction to request from PI the current
      position.
C
      CODE    =       RQCPOS
      WCNT    =       1
      STATUS  =
     ISPFNW( CODE , CHAN , WCNT , REQBUF , BLOCK )
      IF              ( STATUS .NE. 0 ) GOTO 920
C
C     Read the current position.
C
      CODE    =       RDCPOS
      WCNT    =       3
      STATUS  =
     ISPFNW( CODE , CHAN , WCNT , REPBUF , BLOCK )
      IF              ( STATUS .NE. 0 ) GOTO 930
C
C     User program continues from here...
C
      .
      .
      .
C
C     Close the channel.
```

```
C
        STATUS  =       ICLOSE( CHAN )
        IF              ( STATUS .NE. 0 ) GOTO 940
C
C       Return the channel to RT-11.
C
        STATUS  =       IFREEC( CHAN )
        IF              ( STATUS .NE. 0 ) GOTO 950
C
C       Go to common exit.
C
        GOTO            1000
C
C       Error messages begin.
C
  900   TYPE            1
    1   FORMAT          ( 1X , 'No channels available.' )
        GOTO            1000
C
  910   TYPE            2
    2   FORMAT          ( 1X , 'Lookup error on PI:.' )
        GOTO            1000
C
  920   TYPE            3
    3   FORMAT          ( 1X , 'Error requesting current
                         position.' )
        GOTO            1000
C
  930   TYPE            4
    4   FORMAT          ( 1X , 'Error reading current
                         position.' )
        GOTO            1000
C
  940   TYPE            5
    5   FORMAT          ( 1X , 'FATAL - SYSTEM ERROR.' )
        GOTO            1000
C
  950   TYPE            6 , CHAN
    6   FORMAT          ( 1X , 'Channel ' I2 ,
    1                          ' is not currently allocated.' )
C
C       Common Exit point.
C
 1000   CALL            EXIT
C
C       End of sample FORTRAN program for PRO/GIDIS.
C
        END
```

5.4 RESTRICTIONS

Observe the following restrictions when running PRO/GIDIS under RT-11:

- Run PRO/GIDIS only under the XM monitor.
- Run PRO/GIDIS only as the foreground job using the FRUN command.
- The area operation instruction PRINT_SCREEN is not supported.
- VT102 emulation is not supported.

CHAPTER 6

PRO/GIDIS INSTRUCTIONS

This chapter contains detailed reference information for all GIDIS instructions, which are listed in alphabetical order for convenience.

The entry for each GIDIS instruction includes the following information:

- A brief description of the instruction.
- Opcode - used by GIDIS to identify the instruction.
- Length - specifies the number of arguments for the instruction.
- Format - lists and describes each argument.
- Status - indicates conditions for success or failure of the instruction.
- Notes - explain in detail how to use the instruction.
- Device Notes - describe behavior specific to particular GIDIS drivers.
- Example - lists excerpts from a sample MACRO-11 program that uses the instruction.

Unless specified otherwise, all units are in GIDIS Output Space (GOS).

6.1 BEGIN_DEFINE_CHARACTER

BEGIN_DEFINE_CHARACTER starts a character definition block. This causes subsequent instructions to draw into the space associated with the given character, rather than drawing into the entire view surface. This instruction is paired with the END_DEFINE_CHARACTER instruction.

Opcode: 33 **Length:** 4 or 5

Format: BEGIN_DEFINE_CHARACTER char-index, width, nom-width, nom-height, [left-offset]

char-index	The index of the character cell to be loaded. This value must be within the extent of the alphabet (See CREATE_ALPHABET), or -1.
width	This field is used only if the font is defined as variable-width. It then specifies (in GOS units) the implicit movement that should be used for the character being defined. For example, if nom-width were 60, width for *i* would be about 20 and width for *m* would be about 60.
nom-width	Nominal width. The number of GOS units to assign to alphabet width.
nom-height	Nominal height. The number of GOS units to assign to alphabet height.
left-offset	Identifies where the character is placed relative to the current position when it is drawn. Zero, the default, places the left edge of the character at the current position. Values greater than 0 move the cell left; values less than 0 move it right. Units of movement are the same as those for width. Left-offset is specified in GOS units.

Status: SUCCESS if the current alphabet is not equal to 0 and is not a loaded font, char-index is within the extent of the current alphabet, and there are sufficient resources to define this character; otherwise, FAILURE.

Notes:

- Nom-width and nom-height select the natural shape of the character. If the character definition contains a circle, then drawing that character will yield a circle only when unit cell width and height are proportional to nom-width and nom-height.

- Besides affecting shape, nom-width and nom-height control resolution. For example, although 10 x 20 is the same shape as 100 x 200, the latter values give you finer drawing control.

- To define the error character and any undefined character within a font, specify a char-index of -1.

- A character created by a character definition block can be manipulated (for example, scaled and rotated) like any other GIDIS character.

- You cannot use the following instructions inside a character definition block:

```
BEGIN_DEFINE_CHARACTER
LOAD_CHARACTER_CELL
CREATE_ALPHABET
LOAD_BY_NAME
```

- If BEGIN_DEFINE_CHARACTER fails, GIDIS skips all subsequent instructions until it encounters an END_DEFINE_CHARACTER or INITIALIZE. This includes report handling instructions. For example, the following sequence will hang your program:

```
BEGIN_DEFINE_CHARACTER that fails
request report
END_DEFINE_CHARACTER
read report
```

- If left-offset is nonzero, the character should not be drawn in replace, complement negate, or overlay negate modes.

- To abort a character definition, send the INITIALIZE instruction with any argument (including 0). An INITIALIZE 0 instruction aborts a character definition without affecting anything else.

- This instruction implicitly saves all GIDIS attributes for the duration of the BEGIN_DEFINE_CHARACTER process. The END_DEFINE_CHARACTER instruction restores the saved GIDIS attributes. Table 6-1 lists the values in effect during the BEGIN_DEFINE_CHARACTER process.

Table 6-1: Attributes Initialized by BEGIN_DEFINE_CHARACTER

Attribute	Value
output IDS width	nominal width
output IDS height	nominal height
output viewport x origin	0
output viewport y origin	0
output viewport width	nominal width
output viewport height	nominal height
GIDIS output space x origin	0
GIDIS output space y origin	0
GIDIS output space width	nominal width
GIDIS output space height	nominal height
output clipping x origin	0
output clipping y origin	0
output clipping width	nominal width
output clipping height	nominal height
current position x	0
current position y	0
area texture	solid
line texture	solid
logical pixel x offset	0
logical pixel y offset	0
logical pixel width	1 hardware pixel
logical pixel height	1 hardware pixel
cell unit size width	nominal width
cell unit size height	nominal height
cell display size width	nominal width
cell display size height	nominal height
cell movement mode flag	2 (implicit)

Attribute	Value
cell movement mode flag	2 (implicit)
cell explicit movement dx	0
cell explicit movement dy	0
primary color	1
secondary color	0
character cell	all 0's
plane mask	1
writing mode	overlay

Device Notes:

- Plotter GIDIS does not store a character definition as a raster, but rather as a sequence of strokes.
- Except for Plotter GIDIS, a successful CREATE_ALPHABET instruction ensures sufficient resources to store the definition of the character.
- In Video GIDIS, do not allow the terminal subsystem to do a full screen scroll while defining a character.

Example: This illustrates an entire character definitiion.

```
                    ;assume current alphabet is 1, storage
                    ;size of alphabet 1 is 9 by 9.
        .BYTE   4.,33.  ;length = 4,
                        ;opcode for BEGIN_DEFINE_CHARACTER
        .BYTE   3.      ;defining character 3
        .WORD   9.      ;width
        .WORD   90.     ;nom-width
        .WORD   225.    ;nom-height
                        ;now ready to draw into the 9 x 9
                        ;storage area using GOS of
                        ;90 X 225.
        .BYTE   .2,29.  ;length = 2, opcode for SET_POSITION
        .WORD   0.      ;[0,100] is middle of left hand side.
        .WORD   100.
        .BYTE .255.,25. ;introduces uncounted argument list
                        ;opcode for DRAW_LINES
        .WORD   40.     ;[40,200]
```

```
.WORD   200.     ;
.WORD   80.      ;[80,100]
.WORD   100.     ;
.WORD   40.      ;[40,0]
.WORD   0.       ;
.WORD   0.       ;[0,100]
.WORD   100.     ;
.WORD   -32768.  ;end list
                 ;the four lines draw a diamond
.BYTE   0.,36.   ;END_DEFINE_CHARACTER
```

Figure 6-1 illustrates the character defined by this example.

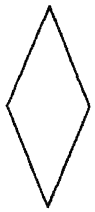


Figure 6-1: Sample Character

6.2 BEGIN_FILLED_FIGURE

BEGIN_FILLED_FIGURE starts the definition of a filled figure. Use DRAW_LINES, DRAW_REL_LINES, DRAW_ARCS, and DRAW_REL_ARCS to enter positions in the filled figure table. Positions are stored in the order given. A corresponding END_FILLED_FIGURE instruction is required to actually fill the figure.

Opcode: 31 **Length:** 0

Format: BEGIN_FILLED_FIGURE

Status: SUCCESS

Notes:

- BEGIN_FILLED_FIGURE sets the filled figure flag to TRUE.
- You should not use the following PRO/GIDIS instructions between a BEGIN_FILLED_FIGURE instruction and its corresponding END_FILLED_FIGURE. (However, this is an unenforced restriction.)

```
BEGIN_FILLED_FIGURE
DRAW_CHARACTERS
DRAW_PACKED_CHARACTERS
SET_GIDIS_OUTPUT_SPACE
SET_OUTPUT_IDS
SET_OUTPUT_VIEWPORT
SET_POSITION
SET_REL_POSITION
```

- The filled figure table must contain at least 1 user-provided point for any drawing to occur. You can enter up to 255 points in the filled figure table. When GIDIS receives the END_FILLED_FIGURE instruction, it adds the original current position twice, as the first and last points of the figure. Thus, GIDIS automatically closes figures for you.
- If you specify too many points, GIDIS uses only the first 255 points. GIDIS ignores points that exceed the capacity of the filled figure table.
- An edge of the filled area is not guaranteed to be identical to a line drawn through the same points, due to differences in drawing direction and round-off errors.

- You may draw lines that cross earlier lines in the filled figure table. Only the enclosed areas will fill. Contrast the examples below. The first creates a square; the second, a bow tie.

- GIDIS attributes used in doing the fill are: primary color, secondary color, writing mode, plane mask, area texture cell, area cell size, and area texture size.

- Complement and complement-negate writing modes can give unexpected results when filled figure areas overlap or abut.

- To abort a filled figure definition, send the INITIALIZE instruction with any argument (including 0). An INITIALIZE 0 instruction aborts a filled figure definition without affecting anything else.

- No drawing is done by the BEGIN_FILLED_FIGURE instruction.

Example:

```
        .BYTE   2.,29.  ;Length=2,opcode for SET_POSITION
        .WORD   100.    ;Current position
        .WORD   100.    ;  now [100,100]
        .BYTE   0.,31.  ;Length=0,opcode for BEGIN_FILLED_FIGURE
                        ;filled figure table now has [100,100]
        .BYTE   6.,26.  ;Length=4,opcode for DRAW_REL_LINES
        .WORD   +100.   ;dx1
        .WORD   +0.     ;dy1
        .WORD   +0.     ;dx2
        .WORD   +100.   ;dy2
        .WORD   -100.   ;dx3
        .WORD   +0.     ;dy3
                        ;Adds points [200,100], [200,200],
                        ;  and [100,200]  to
                        ;  the filled figure table
        .BYTE   0.,32.  ;Length=0,opcode for END_FILLED_FIGURE
                        ;Adds point [100,100] to table
                        ;The area defined by [100,100],
                        ;  [200,100], [200,200], [100,200], and
                        ;  [100,100]--a square--is filled with
                        ;  the current area texture governed by
                        ;  the following writing attributes:
                        ;  writing mode, color map entry,
                        ;  plane mask, primary color,
                        ;  secondary color.
```

Figure 6-2 illustrates the filled figure created by this example.

Figure 6-2: Sample Filled Figure Square

Example:

```
        .BYTE   2.,29.  ;Length=2,opcode for SET_POSITION
        .WORD   100.    ;Current position
        .WORD   100.    ;  now [100,100]
        .BYTE   0.,31.  ;Length=0,opcode for BEGIN_FILLED_FIGURE
                        ;filled figure table now has [100,100]
        .BYTE   6.,26.  ;Length=4,opcode for DRAW_REL_LINES
        .WORD   +100.   ;dx1
        .WORD   +100.   ;dy1
        .WORD   +0.     ;dx2
        .WORD   -100.   ;dy2
        .WORD   -100.   ;dx3
        .WORD   +100.   ;dy3
                        ;Adds points [200,200], [200,100],
                        ;  and [100,200] to
                        ;  the filled figure table

        .BYTE   0.,32.  ;Length=0,opcode for END_FILLED_FIGURE
                        ;Adds point [100,100] to table
                        ;The area defined by [100,100],
                        ;  [200,200], [200,100], [100,200], and
                        ;  [100,100]--a bow tie--is filled with
                        ;  the current area texture governed by
                        ;  the following writing attributes:
                        ;  writing mode, color map entry,
                        ;  plane mask, primary color,
                        ;  secondary color.
```

BEGIN_FILLED_FIGURE

Figure 6-3 illustrates the filled figure created by this example.

Figure 6-3: Sample Filled Figure Bow Tie

6.3 CREATE_ALPHABET

CREATE_ALPHABET reclaims resources used for the current alphabet's font and reserves resources for a new font with the indicated storage size. Storage size in bytes is: 30 + (extent * 2) + (width/8 rounded up) * height * (extent + 1). See Appendix C.

Opcode: 46 **Length:** 4, 5, or 6

Format: CREATE_ALPHABET width, height, extent, flags, [initialize], [ave-width]

width — Is an integer in the range (0 to 64) that specifies the number of horizontal bits in a character pattern.

height — Is an integer in the range (0 to 64) that specifies the number of vertical bits in a character pattern.

extent — Is an integer that specifies the number of characters in the alphabet. Character indices can range from 0 to extent-1, (or 32 to extent +31, if bit 8 of flags is set).

flags — Is a word that specifies one or more of the character renditions and font attributes. Table 6-2 lists the supported renditions.

Table 6-2: CREATE_ALPHABET Flags

Cell Rendition	Bit	Value
Italics	1	2
Bold	3	8
Proportionally spaced	4	16
ASCII	8	256

If ASCII (bit 8) is set, you do not have to include indices 0 through 31 in the font, and the error character is automatically associated with those indices.

initialize — W to initializes all characters in the newly created font. If 0, initialize to blank. If not 0, initialize to solid. If not present, initialize to solid.

ave-width — Is the average width in pixels of glyphs in this font. It is not a true average, but an indication of how many characters fit on an average line of text when this font is used. If not specified, width is used.

Status: SUCCESS if width is 0 to 64, height is 0 to 64, current alphabet number is 1 to 15, extent is greater than or equal to 0, storage size is less than 64 KB, and there are sufficient resources to create the alphabet; otherwise, FAILURE.

Notes:

- To only reclaim the memory used for an alphabet's current font, execute a CREATE_ALPHABET instruction with width, height or extent set to 0.
- If alphabet 15 is current, CREATE_ALPHABET creates a region CRE$AL. See the description of the GIFONT routine in Chapter 4.
- The largest allowable storage size on the Professional is 64KB. If you create a font whose total size is greater than 8KB, the total extent must be less than or equal to 512.
- Specify ave-width for proportionally spaced fonts; width for monospaced fonts.
- When describing a font in an .FDF file (see Appendix D), use ave-width for proportionally spaced fonts and width for monospaced fonts.
- The true limits for font width and height are: (width in bytes) * (height) may not be greater than 512; height may not be greater than 80.
- SET_CELL_UNIT_SIZE uses ave-width, when trying to select the best font.

Device Notes:

- For Plotter GIDIS, no storage is reserved when a CREATE_ALPHABET is done. Space is reserved per character as character definition blocks are processed.

Example:

```
                        ;Current alphabet is alphabet number 2
        .BYTE   4.,46.  ;Length=4, opcode for CREATE_ALPHABET
        .WORD   10.     ;width
        .WORD   16.     ;height
        .WORD   32.     ;extent
        .WORD   8.      ;rend-type bold
                        ;Reclaims space occupied by alphabet 2's
                        ;  current font, allocates space for a
                        ;  new font, and initializes each
                        ;  character in the font to a solid
                        ;  block (because the initialize argument
                        ;  is omitted).
```

6.4 DRAW_ARCS

The DRAW_ARCS instruction draws one or more circular arcs starting from the current position around the specified center(s).

Opcode: 23 **Length:** 3N

Format: DRAW_ARCS x, y, angle

x	Specifies the x coordinate of the arc's center point
y	Specifies the y coordinate of the arc's center point
angle	The angle for the arc is given in degrees, with a positive value meaning counter-clockwise with respect to the view surface. For example, an angle of zero means no drawing is done; +360 or -360 means a full circle is drawn.

Status: SUCCESS if angle is from -360 to +360 and there is no filled figure table overflow; otherwise, FAILURE.

Notes:

- DRAW_ARCS is a repeatable instruction. You can, for example, draw three connected arcs by specifying: x1, y1, angle1, x2, y2, angle2, x3, y3, angle3. The coordinates can be specified either in a counted argument list (with the count supplied in the opcode word), or in an uncounted argument list (with 255 in the opcode word and an END_LIST instruction after the last argument). See END_LIST.

- GIDIS draws an arc as a series of straight lines. The PRO/GIDIS interpreter calculates one line endpoint per 10 degrees of arc (or portion thereof), regardless of the size of the circle.

- If the filled figure flag is TRUE then, instead of drawing the arc, all internally calculated line endpoints are added to the filled figure table.

- The current position is left at the end of the arc, whether the instruction returns SUCCESS or FAILURE.

- Full quadrant arcs (1/4 circle) always end at the exact point expected. Fractional quadrant arcs end at the closest available point. Multiple fractional quadrant arcs are not guaranteed to end at the exact point predicted by your program. For example, a full circle drawn as a 103 degree arc and a 257 degree arc is not guaranteed to leave the current position exactly where it started.

- DRAW_ARCS is affected by the following GIDIS attributes: writing mode, primary color, plane mask, secondary color, pixel size, line texture, and filled figure flag.

- DRAW_ARCS modifies the view surface only inside the clipping rectangle.

Example:

```
                        ;Not in a filled figure definition
                        ;(filled figure flag is FALSE)
                        ;Current position is [500,300]

        .BYTE   3.,23.  ;Length=3, opcode for DRAW_ARCS

        .WORD   400.    :x coordinate of center
        .WORD   300.    ;y coordinate of center
        .WORD   180.    ;180 degrees is one-half a circle
                        ; (counter-clockwise)
                        ;Draws the top half of the circle
                        ;centered at [400,300] with radius 100
                        ;Middle of the arc is [400,200]
                        ;New current position is [300,300]
```

Figure 6-4 shows the arc created by this sample program.

Figure 6-4: Sample Arc

Example:

```
                        ;Not in a filled figure definition
                        ;(filled figure flag is FALSE)
                        ;Current position is [500,300]

        .BYTE   3.,23.  ;Length=3, opcode for DRAW_ARCS

        .WORD   100.    :x coordinate of center
        .WORD   300.    ;y coordinate of center
        .WORD   -90.    ;90 degrees is one-fourth of a circle
                        ; (clockwise)
                        ;Draws a quadrant
                        ;centered at [100,300] with radius 400
                        ;Middle of the arc is [300,400]
                        ;New current position is [100,700]
```

Example:

```
                        ;Inside a filled figure definition
                        ; (filled figure flag is TRUE)
                        ;Current position is [500,300]

        .BYTE   3.,23.  ;Length=3, opcode for DRAW_ARCS

        .WORD   400.    ;Center is [400,300]
        .WORD   300.
        .WORD   -90.    ;90 degrees = 1 quadrant
                        ;Adds eight line endpoints
                        ;(internally calculated)
                        ;plus [400,400] to filled
                        ;figure table
                        ;New current position is [400,400]
```

6.5 DRAW_CHARACTERS

The DRAW_CHARACTERS instruction draws the character identified by the specified character index. The character is taken from the currently selected alphabet.

Opcode: 35 **Length:** N

Format: DRAW_CHARACTERS char-index

char-index Is an unsigned 16-bit word

Status: SUCCESS

Notes:

- DRAW_CHARACTERS is a repeatable instruction. You can, for example, draw several characters in succession by specifying: char-index1, char-index2, char-index3, . . . char-indexn. You can specify characters in either a counted argument list (with the count supplied in the opcode word) or in an uncounted argument list (with 255 in the opcode word and an END_LIST after the last argument.) See END_LIST.

- DRAW_CHARACTERS is affected by several attributes: unit and display cell size, cell slant, cell rotation, rendition mask, current alphabet, writing mode, primary and secondary color, and plane mask.

- If the specified character index is outside the extent of the current alphabet, the error character is drawn. Unless otherwise specified in the font itself, the error character is a checkerboard.

- The current position is updated after a character is drawn according to the cell movement controls. (See the descriptions of the SET_CELL_MOVEMENT_MODE and SET_CELL_EXPLICIT_MOVEMENT instructions.)

- To delete a proportionally spaced character, specify erase writing mode, specify mirrored text by negating the display cell width, and then redraw the character.

- When using local symmetry, the current position after a DRAW_CHARACTERS instruction could be different from that calculated by your program. It is suggested that any series of DRAW_CHARACTERS instructions be followed by a SET_POSITION instruction or a REQUEST_POSITION instruction, unless you do not care exactly where the string ends.

- DRAW_CHARACTERS modifies the view surface only inside the clipping rectangle.
- See also DRAW_PACKED_CHARACTERS.

Example:

```
                        ;Current alphabet = 0 (DEC Multinational)
        .BYTE   3.,35.  ;Length=3, opcode for DRAW_CHARACTERS
        .WORD   65.     ; 'A'
        .WORD   66.     ; 'B'
        .WORD   67.     ; 'C'
                        ; Draws A, B, C from current font for
                        ; alphabet 0
```

Example:

```
                        ;Current alphabet = 1 (user-defined)
        .BYTE   255.,35. ;introduces uncounted argument list
                        ; opcode for DRAW_CHARACTERS
        .WORD   0.
        .WORD   13.
        .WORD   7.
        .WORD   45.
        .WORD   -32768. ;END_LIST
                        ;Draws 4 characters from alphabet 1,
                        ;  which are user-defined characters
```

6.6 DRAW_LINES

The DRAW_LINES instruction draws a straight line from the current position to the specified endpoint. The endpoint is specified as an absolute coordinate pair.

Opcode: 25 **Length:** 2**N**

Format: DRAW_LINES xend, yend

end	Specifies the x coordinate of the line's endpoint
yend	Specifies the y coordinate of the line's endpoint

Status: SUCCESS, provided no filled figure table overflow occurs; on overflow, FAILURE.

Notes:

- The DRAW_LINES instruction is repeatable. You could, for example, draw 3 connected lines by specifying: xend1, yend1, xend2, yend2, xend3, yend3. The coordinates can be specified either in a counted argument list (with the count supplied in the opcode word), or in an uncounted argument list (with 255 in the opcode word and an END_LIST instruction after the last argument). See END_LIST.

- The DRAW_LINES instruction is affected by the following drawing attributes: writing mode, primary color, plane mask, secondary color, pixel size, line texture, and filled figure flag.

- When the filled figure flag is TRUE, this instruction does not draw a line from the current position to the specified point. Instead, it tries to insert [xend, yend] into the filled figure table.

- The current position is updated whether the instruction returns SUCCESS or FAILURE.

- In complement and complement negate modes, the first pixel of a line is skipped and the last pixel is drawn. But if [xend, yend] is itself the current position, the 1 pixel is drawn.

- DRAW_LINES modifies the view surface only inside the clipping rectangle.

Example:

```
                        ;Not in a filled figure definition
                        ;(filled figure flag is FALSE)
                        ;Current position is [200,300]
        .BYTE   2.,25.  ;Length=2, opcode for DRAW_LINES
        .WORD   150.    ;Draw a line from [200,300]
        .WORD   200.    ;to [150,200]
                        ;New current position is [150,200]
```

Example:

```
                        ;current position is [150,200]
                        ;not in a filled figure definition
        .BYTE   4.,25.  ;Length=4, opcode for DRAW_LINES
        .WORD   600.    ;xend1
        .WORD   -10.    ;yend1
        .WORD   300.    ;xend2
        .WORD   +10.    ;yend2
                        ;Draw lines from [150,200] to [600,-10]
                        ;then from [600,-10] to [300,10]
                        ;New current position is [300,10]
                        ;Note that both the -10 and the +10 are
                        ;absolute coordinates.
```

Example:

```
                        ;current position is [300,10]
                        ;not in a filled figure definition
        .BYTE   255.,25.;introduces uncounted argument list
                        ;opcode for  DRAW_LINES
        .WORD   400.    ;xend1
        .WORD   40.     ;yend1
        .WORD   -32768. ;ENDLIST terminator value
                        ;Draws a line from [300,10] to [400,40]
                        ;New current position is [400,40]
```

Example:

```
                        ;Inside a filled figure definition
                        ; (filled figure flag is TRUE)
        .BYTE   4.,25.  ;Length=4, opcode for DRAW_LINES
        .WORD   300.    ;xend1
        .WORD   200.    ;yend1
        .WORD   300.    ;xend2
        .WORD   300.    ;yend2
                        ;The points [300,200] and [300,300] are
                        ;added to the filled figure table
                        ;new current position is [300,300]
```

6.7 DRAW_PACKED_CHARACTERS

DRAW_PACKED_CHARACTERS makes drawing of ASCII strings more efficient, because it enables you to pack two ASCII characters into one word. You can use DRAW_PACKED_CHARACTERS for non-ASCII alphabets, if the indices are less than 255. It uses the low order byte before the high order byte. Otherwise, DRAW_PACKED_CHARACTERS is equivalent to DRAW_CHARACTERS.

Opcode: 74 **Length:** N

Format: DRAW_PACKED_CHARACTERS 2charindex

2charindex is two 8-bit character indices

Status: SUCCESS

Notes:

- DRAW_PACKED_CHARACTERS is appropriate for any alphabet whose extent is less than 255.
- A character index of 255 explicitly performs no operation. Thus, if you want to draw 1 character, place 255 in the high order byte of the argument.
- Using a DRAW_PACKED_CHARACTERS instruction with repeated arguments is the fastest way to draw a long string with GIDIS.
- DRAW_PACKED_CHARACTERS is a repeatable instruction. Characters can be specified either in a counted argument list (with the count supplied in the opcode word) or in an uncounted argument list (with 255 in the opcode word and an END_LIST after the last argument).
- The current position is updated after a character is drawn, according to the cell movement controls. (See SET_CELL_MOVEMENT_MODE and SET_CELL_EXPLICIT_MOVEMENT.)
- To delete a proportionally spaced character, specify erase writing mode, specify mirrored text by negating display cell width, and then redraw the character.
- When using local symmetry, the current position after a DRAW_PACKED_CHARACTERS instruction could be different from that calculated by your program. It is suggested that any series of DRAW_PACKED_CHARACTERS instructions be followed by

a SET_POSITION instruction or a REQUEST_POSITION instruction, unless you do not care exactly where the string ends.

- The DRAW_PACKED_CHARACTERS instruction is affected by several attributes: unit and display size, cell slant, cell rotation, rendition mask, current alphabet, writing mode, primary and secondary color, and plane mask.

- If the specified character index is outside the extent of the current alphabet, the error character is drawn. Unless otherwise specified in the font itself, the error character is a checkerboard.

- DRAW_CHARACTERS modifies the view surface only inside the clipping rectangle.

Example:

```
                        ;assume current alphabet is 0
        .BYTE   3.,74.  ;length=3 words,opcode for
                        ;DRAW_PACKED_CHARACTERS
        .BYTE   116.,101. ;'t', 'e'
        .BYTE   115.,116. ;'s', 't'
        .BYTE   49.,255.  ; '1', "no character"
                        ;draws the string "test1"
```

Example:

```
        .BYTE  1.,38.     ;length=1, opcode for SET_ALPHABET
        .WORD  1.         ;alphabet 1
        .BYTE  255.,74.   ;introduces uncounted argument list
                          ;opcode for DRAW_PACKED_CHARACTERS
        .BYTE  0.,1.      ;draw characters 0,1
        .WORD -32768.     ;draws characters that you defined in
                          ;index 0 and 1 of alphabet 1
```

6.8 DRAW_REL_ARCS

DRAW_REL_ARCS draws a circular arc from the current position around the specified center.

Opcode: 27 **Length:** 3N

Format: DRAW_REL_ARCS dx, dy, angle

dx	Specifies the x coordinate of the arc's center point as: x of current position + dx
dy	Specifies the y coordinate of the arc's center point as: y of current position + dy
angle	The angle for the arc is given in degrees, with a positive value meaning counter-clockwise with respect to the view surface. An angle of zero means no drawing is done; +360 or -360 means a full circle is drawn.

Status: SUCCESS, if angle is within a range of -360 to +360 and there is no filled figure table overflow or arithmetic overflow; otherwise, FAILURE.

Notes:

- An arc is drawn as a series of straight lines. The PRO/GIDIS interpreter calculates one line endpoint per 10 degrees of arc (or portion thereof), regardless of the size of the circle.
- If the filled figure flag is TRUE, instead of drawing the arc, all internally calculated line endpoints are added to the filled figure table.
- DRAW_REL_ARCS is a repeatable instruction. You can, for example, draw three connected arcs by specifying: dx1, dy1, angle1, dx2, dy2, angle2, dx3, dy3, angle3. The coordinates can be specified either in a counted argument list (with the count supplied in the opcode word), or in an uncounted argument list (with 255 in the opcode word and an END_LIST instruction after the last argument). See END_LIST.
- The current position is left at the end of the last arc.

- Full quadrant arcs (1/4 circle) always end at the exact point expected. Fractional quadrant arcs end at the closest available point. Multiple fractional quadrant arcs are not guaranteed to end at the exact point predicted by your program. For example, a full circle drawn as a 103 degree arc and a 257 degree arc is not guaranteed to leave the current position exactly where it started.

- DRAW_REL_ARCS is affected by the following GIDIS attributes: writing mode, primary color, plane mask, secondary color, pixel size, line texture, and filled figure flag.

- DRAW_REL_ARCS modifies the view surface only inside the clipping rectangle.

Example:

```
                        ;Current position is [400,300]
                        ;filled figure flag is FALSE
        .BYTE   3.,27.  ;Length=3,opcode for DRAW_REL_ARCS
        .WORD   -100.   ;Center is [-100,+30]
        .WORD   +30.    ;Relative to current position
        .WORD   -90.    ;90 degrees = one quadrant (clockwise)
                        ;Draws one quadrant from [400,300] to
                        ;[330,430] centered at [300,330]
                        ;New current position is [330,430]
```

Example:

```
                        ;Current position is [330,430]
                        ;filled figure flag is FALSE
        .BYTE   6.,27.  ;Length=3, opcode for DRAW_REL_ARCS
        .WORD     +35.  ;
        .WORD     -50.  ;Center is [+35,-50]
        .WORD      90.  ;[365,380], 90 degree arc
        .WORD     -35.  ;Current position is now [415,415]
        .WORD     +50.  ;Center is 380,465]
        .WORD      90.  ;90 degrees
                        ;draws a lens shaped object with two
                        ;circular arcs.
```

6.9 DRAW_REL_LINES

The DRAW_REL_LINES instruction draws a straight line from the current position to the specified endpoint. The endpoint coordinates are specified relative to the current position.

Opcode: 26 **Length:** 2N

Format: DRAW_REL_LINES dxend, dyend

dxend	Specifies the x coordinate of the line's endpoint as: current position + dxend
dyend	Specifies the y coordinate of the line's endpoint as: current position + dyend

Status: SUCCESS, if no last pair arithmetic overflow or filled figure table overflow occurs; on overflow, FAILURE. On success, the current position is set to [x of current position + dxend, y of current position + dyend]. On failure, the current position is not changed.

Notes:

- The DRAW_REL_LINES instruction is repeatable. You can,for example, draw 3 connected lines by specifying: dxend1, dyend1, dxend2, dyend2, dxend3, dyend3. The coordinates can be specified either in a counted argument list (with the count supplied in the opcode word), or in an uncounted argument list (with 255 in the opcode word and an END_LIST instruction after the last argument). See END_LIST.

- The DRAW_REL_LINES instruction is affected by the following drawing attributes: writing mode, primary color, plane mask, secondary color, pixel size, line texture, and filled figure flag.

- When the filled figure flag is TRUE, this instruction does not draw a straight line from the current position to the specified point. Instead, it tries to insert [x of current position + dxend, y of current position + dyend] into the filled figure table. No drawing occurs until the END_FILLED_FIGURE instruction is processed.

- In complement and complement negate mode, the first pixel of a line is skipped and the last pixel is drawn. But if [x of current position + dyend, y of current position + dyend] is itself the current position, the one pixel is drawn.

- DRAW_REL_LINES modifies the view surface only inside the clipping rectangle.

Example:

```
                        ;Not in a filled figure definition
                        ;(filled figure flag is FALSE)
                        ;Current position is [100,100]
        .BYTE   4.,26.  ;Length=4, opcode for DRAW_REL_LINES
        .WORD   +10.    :dxend1
        .WORD   -10.    ;dyend1
        .WORD   +30.    ;dxend2
        .WORD   +15     ;dyend2
                        ;Draw lines from [100,100] to [110,90]
                        ;and from [110,90] to [140,105]
                        ;New current position is [140,105]
```

Example:

```
                        ;Current position is [140,105]
                        ;not in a filled figure definition
        .BYTE   255.,26. ;introduces uncounted argument list
                        ;opcode for DRAW_REL_LINES
        .WORD   10.     ;dxend1
        .WORD   -30.    ;dyend1
        .WORD   20.     ;dxend2
        .WORD   +60.    ;dyend2
        .WORD   -32768. ;END_LIST
                        ;Draw line from [140,105] to [150,75]
                        ;and then to [170,135]
                        ;New current position is [170,135]
```

Example:

```
                        ;Inside a filled figure definition
                        ; (filled figure flag is TRUE)
                        ;Current position is [100,100]
        .BYTE   5.,26.  ;Length=5, opcode for DRAW_REL_LINES
        .WORD   100.    ;dxend1
        .WORD   0.      ;dyend1
        .WORD   0.      ;dxend2
        .WORD   100.    ;dyend2
                        ;Adds the points [200,100] and [200,200]
                        ;to the filled figure table
                        ;New current position is [200,200]
```

6.10 END_DEFINE_CHARACTER

END_DEFINE_CHARACTER terminates a character definition block and restores the GIDIS attributes saved by the BEGIN_DEFINE_CHARACTER instruction.

Opcode: 36 **Length:** 0

Format: END_DEFINE_CHARACTER

Status: SUCCESS if character definition flag is TRUE; otherwise, FAILURE.

Notes:

- The defined character can now be used like any other character in DRAW_CHARACTERS and DRAW_PACKED_CHARACTERS.

Device Notes:

- In Video GIDIS, while you are defining a large character, all but its bottom 16 lines (32 in high resolution mode on the Professional 380) are visible at the bottom of the screen. When END_DEFINE_CHARACTER is processed, the area occupied by the character is set to current secondary color.

Example: See BEGIN_DEFINE_CHARACTER.

6.11 END_FILLED_FIGURE

END_FILLED_FIGURE terminates the definition of a closed figure, and fills the figure. This instruction is used in conjunction with the BEGIN_FILLED_FIGURE instruction.

Opcode: 32 **Length:** 0

Format: END_FILLED_FIGURE

Status: SUCCESS if there is at least one point in the filled figure table; otherwise, FAILURE.

Notes:

- The filled figure table must contain at least 1 user-provided point for any drawing to occur. GIDIS provides the initial current position twice, at the beginning and end, thereby automatically closing the figure.
- If you specify too many points, GIDIS uses only the first 255 points, and draws a straight line connecting the 255th point with the initial current position. (255 is the maximum number of user-provided points in the filled figure table.)
- The current position is unchanged by END_FILLED_FIGURE. The current position remains wherever the last drawing instruction in the figure block set it.
- END_FILLED_FIGURE turns off the filled figure flag.
- This instruction modifies the view surface only inside the clipping rectangle.

Example: See BEGIN_FILLED_FIGURE

6.12 END_LIST

END_LIST indicates the end of an uncounted argument list. This instruction follows the last argument in the list. The PRO/GIDIS instructions often used with an uncounted argument list are: DRAW_LINES, DRAW_REL_LINES, DRAW_ARCS, DRAW_REL_ARCS, DRAW_CHARACTERS, DRAW_PACKED_CHARACTERS.

Opcode: 128 **Length:** must be 0

Format: END_LIST

Status: SUCCESS

Notes:

- You specify an uncounted argument list by placing a length of 255 in an instruction's opcode word.
- 128 * 256 + 0 equals -32768. Thus, -32768 may not be the value of an argument word in an uncounted argument list. However, -32768 is valid as an argument in a counted argument list. For example, the point [-32768,0] could not be sent in a DRAW_LINES instruction terminated by an END_LIST instruction, but could be sent in a DRAW_LINES instruction with counted arguments.

Example:

```
        .BYTE   255.,25. ;length=255 is a special value that
                         ;  does not indicate 255 data words
                         ;  following, but that there are an
                         ;  unknown number of words
                         ;  following, to be terminated
                         ;  with the END_LIST instruction.
                         ;opcode for DRAW_LINES
        .WORD   100.     ;DRAW_LINES data
        .WORD   110.     ;DRAW_LINES data
        .WORD   -32768.  ;END_LIST
```

6.13 END_PICTURE

END_PICTURE logically terminates the current picture. The action performed by END_PICTURE depends on the current device.

Opcode: 24 **Length:** 0

Format: END_PICTURE

Status: SUCCESS

Notes:

- It is recommended that you use NEW_PICTURE and END_PICTURE to enclose the instructions used in drawing a picture.
- END_PICTURE simulates a FLUSH_BUFFER instruction.

Device Notes:

- For a GIDIS that builds a virtual bitmap (for example, Palette GIDIS), END_PICTURE causes the bitmap to be output to the device.
- For Sixel GIDIS, an END_PICTURE does a formfeed. However if you are using the VDM interpreter to print the picture as part of a document, the formfeed is suppressed.
- For Palette GIDIS, an END_PICTURE advances the film, provided you are not passing the picture through the VDM interpreter.
- For Plotter GIDIS, an END_PICTURE advances the paper (or ejects the paper in the case of single sheet feed), provided you are not passing the picture through the VDM interpreter.

Example:

```
        .BYTE   0.,6.           ;length=0,opcode for NEW_PICTURE
          .                     ;
          .                     ;drawing instructions
          .                     ;
        .BYTE   0.,24.          ;length=0,opcode for END_PICTURE
          .                     ;
          .                     ;wait for operator response
          .                     ;perhaps
        .BYTE   0.,6.           ;length=0,opcode for NEW_PICTURE
          .                     ;
          .                     ;more drawing instructions
          .                     ;
```

6.14 ERASE_CLIPPING_REGION

ERASE_CLIPPING_REGION sets every pixel inside the current clipping rectangle to the current secondary color. This instruction provides a way to clear an area without implying the beginning of a new picture.

Opcode: 48 **Length:** 0

Format: ERASE_CLIPPING_REGION

Status: SUCCESS

Notes:

- Do not use this instruction as a substitute for NEW_PICTURE and END_PICTURE.
- You should use ERASE_CLIPPING_REGION, rather than BEGIN_FILLED_FIGURE and END_FILLED_FIGURE to clear a rectangular area of the view surface.
- The current writing mode, current area texture, and primary color do not affect this instruction. However, plane mask does.

Device Notes:

- Plotter GIDIS ignores ERASE_CLIPPING_REGION.

Example:

```
        .BYTE   0.,48.  ;Length=0,
                        ;opcode for ERASE_CLIPPING_REGION
```

6.15 FLUSH_BUFFER

FLUSH_BUFFER forces execution of any pending GIDIS processing.

Opcode: 28 **Length:** 0

Format: FLUSH_BUFFER

Status: SUCCESS

Notes:

- FLUSH_BUFFER enables you to ensure that all previous drawing instructions have been executed prior to requesting operator response or the like.

Example:

```
        .BYTE   0.,28.  ;length=0,opcode for FLUSH_BUFFER
```

6.16 INITIALIZE

INITIALIZE restores PRO/GIDIS subsystems to their default states. Also, if a character definition block or filled figure block is active, INITIALIZE aborts it.

Opcode: 1 **Length:** 1

Format: INITIALIZE sub-mask

sub-mask	Is a word that specifies zero or more of PRO/GIDIS's subsystems. The subsystems defined at this time are listed in Table 6-3. A subsystem is represented in the mask value as a bit, as shown in the table. For example, a value of 6 (bit 2 + bit 1) resets text and writing attributes.

Status: SUCCESS

Table 6-3: Initialization of Subsystems

Subsystem	Description	Bit	Value
Addressing	Sets IDS, Viewport, GOS and clipping region to 960 x 600. Also sets all attributes that specify distances or coordinates (for example, unit cell size).	0	1
Writing Attributes	Reinitializes writing mode, primary color, secondary color, line and area texture, planes selected, and pixel size.	1	2

Subsystem	Description	Bit	Value
Text	Resets the current alphabet, unit size, display size, cell rotation, cell rendition, implicit cell movement flag, and explicit cell movement.	2	4
Color Map	Reinitializes the color map.	4	16
Alphabet	Clears all user-defined alphabets and sets family ID of alphabet 0 to "DGIDIS".	5	32
Cursor	Resets the output cursor and output rubber band.	8	256

Notes:

- INITIALIZE 0 is useful, as it aborts any blocks begun with BEGIN_FILLED_FIGURE and BEGIN_DEFINE_CHARACTER without affecting any GIDIS subsystems.
- You can combine mask bits to initialize multiple subsystems in one instruction.
- A mask of -1 decimal (177777 octal) initializes all subsystems.
- The order of initialization is: (1) addressing, (2) writing attributes, (3) text, (4) color map, (5) alphabet storage, and (6) cursor.
- .GID files that use default text attributes may not come out as expected, because some defaults are appropriate only for Video GIDIS.
- Table 6-4 lists all of the GIDIS attributes affected and their values after initialization. Note that some attributes are included in more than one subsystem. All coordinates and distances are in GOS, unless otherwise noted.

Table 6-4: Values of GIDIS Attributes After an INITIALIZE

Attribute	Value
Addressing Subsystem	
output IDS width	960
output IDS height	600
output viewport x origin	0 in IDS
output viewport y origin	0 in IDS
output viewport width	960 in IDS
output viewport height	600 in IDS
GIDIS output space x origin	0
GIDIS output space y origin	0
GIDIS output space width	960
GIDIS output space height	600
output clipping x origin	0
output clipping y origin	0
output clipping width	960
output clipping height	600
current position x	0
current position y	0
line texture size	N/A
area texture width	12
area texture height	25
logical pixel width	0 (1 hardware pixel)
logical pixel height	0 (1 hardware pixel)
logical pixel x offset	0
logical pixel y offset	0
cell movement mode flag	2 (implicit)
cell explicit movement dx	0
cell explicit movement dy	0
cell display size width	12
cell display size height	25
cell unit size width	12
cell unit size height	25

Attribute	Value
Writing Attributes Subsystem	
primary color	7
secondary color	0
plane mask	all available planes
writing mode	overlay
logical pixel width	0
logical pixel height	0
logical pixel x offset	0
logical pixel y offset	0
line texture pattern	solid (all ones)
line texture length	N/A
line texture size	N/A
area texture alphabet	-1
area texture character	0 (solid)
area texture width	12
area texture height	25
Text Subsystem	
current alphabet	0
cell display size width	12
cell display size height	25
cell unit size width	12
cell unit size height	25
cell rotation	0
cell oblique	0
cell rendition	0
cell movement mode flag	2 (implicit)
cell explicit movement dx	0
cell explicit movement dy	0

Attribute	Value
Color Map Subsystem (values associated)	

	R	*G*	*B*	*M*	*Color*	*Mono*
color map [0]	.0	.0	.0	.0	black	(dark)
color map [1]	.2	.2	.6	.2	blue	(dk. gray)
color map [2]	.7	.2	.2	.3	red	(lt. gray)
color map [3]	.2	.7	.2	.4	green	(light)
color map [4]	.6	.6	.6	.7	white	(light)
color map [5]	.6	.6	.6	.7	white	(light)
color map [6]	.6	.6	.6	.7	white	(light)
color map [7]	.6	.6	.6	.7	white	(light)

Alphabet Storage Subsystem	
family name of alphabet 0	"DGIDIS"
Cursor Subsystem	
output cursor alphabet	-1
output cursor character index	N/A
output cursor width	N/A
output cursor height	N/A
output cursor x offset	N/A
output cursor y offset	N/A
output cursor rendition	blinking
output rubber band type	none

Example:

```
        .BYTE   1.,1.           ;length=1,opcode for INITIALIZE
        .WORD   1.!2.!4.        ;addressing, writing attributes,
                                ;and text subsystems mask bits
                                ;are set
```

6.17 LOAD_BY_NAME(1)

LOAD_BY_NAME(1) loads a pre-built font into the current alphabet. The argument list is a pair of words which contain a region name in Radix-50.

Opcode: 37 **Length:** 2

Format: LOAD_BY_NAME(1) name-0, name-1

name-0	3 radix-50 characters
name-1	3 radix-50 characters

Status: SUCCESS if the region named identifies a valid region, the region has the proper format, the current alphabet number is not 0, and there are sufficient resources to load the font region; otherwise, FAILURE.

Notes:

- Subsequent SET_ALPHABET instructions do not affect previous LOAD_BY_NAME INSTRUCTIONS. For example, if you loaded font MYALPH into alphabet 1, it would remain alphabet 1's font until INITIALIZE 32, another LOAD_BY_NAME, or a CREATE_ALPHABET was processed for alphabet 1.

- If no such region can be found or installed, GIDIS simulates a LOAD_BY_NAME(2) and loads "DGIDIS", the default family ID for alphabet 0.

- A GIDIS font file is an RSX Common Library. In other words, installing a GIDIS font file creates a font region.

- A font region must conform to the format shown in Appendix C.

- A font region can be accessed in one of 3 ways:

  1. Prior to doing the LOAD_BY_NAME(1), you can create and load the region in your application.

  2. Prior to doing the LOAD_BY_NAME(1), you can install a font file with the DCL INSTALL command, PROTSK, or your application installation file (.INS).

  3. You can rely on GIDIS to install the region's font file when the LOAD_BY_NAME(1) is done. You enable GIDIS to install the file in either of two ways:

1. Place the font file on LB:[ZZFONT] and name it region-name.TSK. (Note: $'s and .'s in a region name become Z's in the filename).

2. Describe the font in an .FDF file. See Appendix D.

Example:

```
.BYTE   2.,37.  ;length=2, opcode for LOAD_BY_NAME
.Radix-50 "BOLD";let MACRO-11 compute the Radix-50 for
 BOLD
```

Example:

```
.BYTE   2.,37.                  ;Radix-50 for MYALPH
.WORD   050500+001750+000001    ;MYA
.WORD   045400+001200+000010    ;LPH
```

6.18 LOAD_BY_NAME(2)

LOAD_BY_NAME(2) associates the current alphabet with the specified font family. When a subsequent DRAW_CHARACTERS or DRAW_PACKED_CHARACTERS is done, GIDIS finds the font file in the family that best matches the current GIDIS text attributes. See Appendix D.

Opcode: 37 **Length:** 3 to 7

Format: LOAD_BY_NAME(2) Ch1, Ch2, Ch3, ...Chn

Ch1 - Chn Is a font family ID, encoded as 1 character per word

Status: SUCCESS

Notes:

- Subsequent SET_ALPHABET instructions do not affect previous LOAD_BY_NAME instructions. For example, if you loaded family ID MYALPH into alphabet 1, it would remain alphabet 1's family ID until INITIALIZE 32, another LOAD_BY_NAME, or a CREATE_ALPHABET was processed for alphabet 1.
- The default family ID for alphabet 0 is DGIDIS.
- Family IDs are mapped to uppercase. For example, specifying "dgidis" is equivalent to specifying "DGIDIS".
- A font must be described in an .FDF file on LB:[ZZFONT] to be accessible via a LOAD_BY_NAME(2) (see Appendix D).
- If the specified family has no members, GIDIS simulates a LOAD_BY_NAME(2) "DGIDIS."

Example:

```
        .BYTE   1.,38.  ;Length=1,opcode for SET_ALPHABET
        .WORD   1.      ;Selects alphabet 1 as current alphabet
        .BYTE   6.,37.  ;Length=6,opcode for LOAD_BY_NAME
        .WORD   68.     ;D
        .WORD   71.     ;G
        .WORD   73.     ;I
        .WORD   68.     ;D
        .WORD   73.     ;I
        .WORD   83.     ;S,associates alphabet 1 with family
                        ;ID "DGIDIS"
```

6.19 LOAD_CHARACTER_CELL

LOAD_CHARACTER_CELL defines a character cell from the specified data. This instruction acts on the current alphabet.

Opcode: 34 **Length:** 2 + N

Format: LOAD_CHARACTER_CELL char-index, width, d0, d1,...,dn

char-index	The index of the character cell to be loaded. This value must be in the range 0 to extent-1, where extent is the total character count for the current alphabet.
width	The width value must be in the range 0 to the width value given with the CREATE_ALPHABET instruction that established the alphabet.
d0, d1,...dn	Zero or more words of data to be loaded into the character cell. The top character cell row is loaded from the first data word(s), the second row from the next data words, and so forth. Excess data words are ignored, and missing data words are assumed to be 0's. Each row of the cell is (alphabet width + 15)/16 data words. For example, an 8-bit wide alphabet has 1 word per row, and a 20-bit wide alphabet has 2 words per row.

Status: SUCCESS if not within a character definition block (See BEGIN_DEFINE_CHARACTER), character index is in range (see CREATE_ALPHABET), width is in the range 0 to alphabet width, and a CREATE_ALPHABET has been done for the current alphabet; otherwise, FAILURE.

Notes:

- The defined character can now be used like any other character in SET_AREA_TEXTURE, SET_OUTPUT_CURSOR, DRAW_CHARACTER and DRAW_PACKED_CHARACTER instructions.
- The leftmost pixel in a row comes from the low-order bit in the appropriate data word.

Example:

```
                    ;Alphabet 2 has width of 5, height of 6,
                    ;  and extent of 10

        .BYTE   7.,34.  ;Length=7, opcode for LOAD_CHARACTER_CELL

        .WORD   9.      ;Character index (last cell in alphabet)
        .WORD   5.      ;Width
        .WORD   ^B00001 ;Pattern:        ON   --   --   --   --
        .WORD   ^B00011 ;                ON   ON   --   --   --
        .WORD   ^B00101 ; (Note the      ON   --   ON   --   --
        .WORD   ^B01001 ; bit reversal)  ON   --   --   ON   --
        .WORD   ^B11111 ;                ON   ON   ON   ON   ON
                        ;                --   --   --   --   --
                        ;Last row not given; set to 0's
                        ;automatically.
                        ;Character is a triangle
```

6.20 NEW_PICTURE

NEW_PICTURE indicates the beginning of a new picture.

Opcode: 6 **Length:** 0

Format: NEW_PICTURE

Status: SUCCESS

Notes:

- It is recommended that you use NEW_PICTURE and END_PICTURE to enclose the instructions used in drawing a picture.
- Secondary color is written to the view surface subject to the plane mask in effect at the time NEW_PICTURE executes. (See SET_PLANE_MASK.)
- A NEW_PICTURE clears all of hardware address space, regardless of the current clipping region. In particular, it clears the 32-pixel bands on both sides of the screen not normally used in Video GIDIS.

Device Notes:

o NEW_PICTURE does not affect picture background in Plotter GIDIS.

Example:

```
        .BYTE   0.,6.           ;length=0,opcode for NEW_PICTURE
```

6.21 NOP

NOP performs no operation. Execution of a NOP has no effect on the current state of PRO/GIDIS, other than to set the status flag to SUCCESS.

Opcode: 0 **Length:** 0

Format: NOP

Status: SUCCESS

Note:

- This instruction is useful for transparently inserting information from a higher level protocol into a stream of GIDIS instructions. Use a nonzero length, when you want to insert information.

Example:

```
        .BYTE   0.,0.   ;length=0,opcode for NOP
```

Example:

```
        .BYTE   2.,0.   ;length=2,opcode for NOP
        .WORD   1540.   ;private data (ignored by PRO/GIDIS)
        .WORD   71.     ;private data (ignored by PRO/GIDIS)
```

6.22 PRINT_SCREEN

PRINT_SCREEN sends the specified portion of the video bitmap to a sixel printer connected to the printer port.

Opcode: 141 **Length:** 6 or 7

Format: PRINT_SCREEN x, y, width, height, hxly, dxly, [mask]

x	Specifies the leftmost horizontal coordinate of the GOS data to be printed
y	Specifies the uppermost vertical coordinate of the GOS data to be printed
width	Width of the area to be printed
height	Height of the area to be printed
hxly	Specifies the horizontal offset from the current printhead location to where you want to begin printing the screen data.
dxly	Specifies the vertical offset from the current printhead location to where you want to begin printing the screen data.
mask	Specifies the color indexes that cause printing a dot on the paper. The low order bit is color 0, the next bit color 1, and so on. If mask is omitted, it is generated as follows. In a single plane system (no EBO), a pixel value of 0 is mapped to a skip (leaves paper white) and a 1 is mapped to a strike (prints on the paper). On multi-plane systems, the value of the color map is tested as follows. If the entry (color index of point) equals 0, the point is skipped (white). If not 0, the point prints.

Status: SUCCESS

Notes:

- Applies to Video GIDIS only.
- If the printer port does not have a sixel printer connected, nothing occurs.

Example:

```
        .BYTE   6.,141. ;Length=6, opcode for PRINT_SCREEN
        .WORD   100.    ;Upper left bitmap corner
        .WORD   100.    ;   is [100,100]
        .WORD   400.    ;Data to be printed is 400 units wide
        .WORD   200.    ;   by 200 units high
        .WORD   0.      ;Begin printing at current printhead
        .WORD   0.      ;   location
```

6.23 REQUEST_CELL_STANDARD

REQUEST_CELL_STANDARD reports the GOS dimensions you would have to specify to generate a standard size character. A standard size character has dimensions such that when its width/height = 8/5, 80 characters fit across the device and 24 lines fit vertically.

Opcode: 54 **Length:** 0

Format: REQUEST_CELL_STANDARD

Status: SUCCESS

The report consists of 5 words:

Report Header, unit-wd, unit-ht, display-wd, display-ht

where

unit-wd	Is the unit cell width of the standard size character.
unit-ht	Is the unit cell height of the standard size character.
display-wd	Is normally the same as unit-wd. However if the current alphabet is 0, this value is 11/12 of the current cell width.
display-ht	Is normally the same as unit-ht. However if the current alphabet is 0, this value is 11/12 of the current cell height.

Notes:

- This instruction takes into account the storage size of the current alphabet and the character rotation currently in effect. As a result, the standard size for alphabet 0 (DEC Multinational) is not necessarily the same as the standard size for a user alphabet.
- Rounding could take place converting from device coordinates to GIDIS space. If your program later sets unit cell size to 'n' times the size of the standard, the characters actually formed might not be precisely 'n' times the standard.

Example:

```
        .BYTE   0.,54.  ;Length=0,
                        ;opcode for REQUEST_CELL_STANDARD
                        ; Byte  4.   (Data words following)
                        ; Byte  5.   (Cell Standard Rpt. Tag)
                        ; Word  9.   (Unit-wd)
                        ; Word  20.  (Unit-ht)
                        ; Word  8.   (Display-wd)
                        ; Word  20.  (Display-ht)
                        ;
```

6.24 REQUEST_CURRENT_POSITION

REQUEST_CURRENT_POSITION reports the absolute location of the current position in GIDIS Output Space. The current position is the display location at which the next character, line, or arc would be drawn.

Opcode: 55 **Length:** 0

Format: REQUEST_CURRENT_POSITION

Status: SUCCESS

The report consists of 3 words:

Report header, current x, current y

Notes:

- The current position is not necessarily the same as the last position given to SET_POSITION or DRAW_LINES; DRAW_CHARACTERS and DRAW_ARCS instructions also move the current position.
- REQUEST_CURRENT_POSITION is most useful following a DRAW_ARCS or a DRAW_CHARACTERS (local symmetry), since your program cannot determine precisely where PRO/GIDIS leaves the current position after these instructions.

Example:

```
        .BYTE   0.,55.  ;Length=0,
                        ;opcode for REQUEST_CURRENT_POSITION
                        ;This instruction causes the following
                        ;  report to be placed in the report
                        ;  queue if there is sufficient room.
                        ;
                        ; Byte  2. Data words following
                        ; Byte  1. Current Position Report Tag
                        ; Word  x  PRO/GIDIS coordinates
                        ; Word  y  for the current position
```

6.25 REQUEST_OUTPUT_SIZE

REQUEST_OUTPUT_SIZE reports the attributes of the current device's view surface.

Opcode: 57 **Length:** 0

Format: REQUEST_OUTPUT_SIZE

Status: SUCCESS

The report consists of 10 words:
Report header, ulx, uly, screen_width, screen_height, total_width, total_height, resolution_x, resolution_y, Total_plane_mask

where

[ulx, uly]	Are the coordinates of the upper left corner of the device's view surface in IDS units
Screen_width	Is device width in IDS units
Screen_height	Is device height in IDS units
Total_width	Is device width in IDS units
Total_height	Is device height in IDS units
Resolution_x	Is device width in HAS x units
Resolution_y	Is device height in HAS y units
Total_plane_mask	Is the plane mask that contains a 1 for every plane accessible. See device notes of SET_PLANE_MASK.

Example:

```
                    ;Assume PRO 350 Video with EBO
                    ;Assume IDS is 960 by 600
        .BYTE   0.,57.  ;length=0,opcode for REQUEST_OUTPUT_SIZE
                    ;
                    ;BYTE 9.    9 words following output size
                    ;           report tag
                    ;BYTE 2.    OUTPUT_SIZE_REPORT tag
                    ;WORD -32.  IDS coordinate of device's
                    ;WORD 0.    upper left corner is [-32,0]
                    ;WORD 1024. IDS width and height of
                    ;WORD 600.  entire view surface
                    ;WORD 1024. IDS width and height of
                    ;           entire view surface
                    ;WORD 600.
                    ;1024.      number of pixels in total
                    ;           device width
                    ;WORD 240.  number of pixels in total
                    ;           device height
                    ;WORD 7.    total plane mask
```

6.26 REQUEST_STATUS

REQUEST_STATUS reports the success or failure of the last PRO/GIDIS instruction. All PRO/GIDIS instructions set the status variable.

Opcode: 58 **Length:** 0

Format: REQUEST_STATUS

Status: SUCCESS

The report consists of 2 words:

Report header, status

where the low-order bit of the status word is either

1 - indicating SUCCESS

0 - indicating FAILURE.

Notes:

- No other codes are defined. (Codes other than 0 or 1 are reserved for future use.)
- FAILURE status is not saved. If your program needs information about the success or failure of every instruction, you must place a REQUEST_STATUS instruction after each PRO/GIDIS instruction.
- Testing is recommended only following major PRO/GIDIS instructions, such as CREATE_ALPHABET.

Example:

```
                    ;assumes previous instruction failed
        .BYTE   0.,58.  ;Length=0,
                        ;opcode for REQUEST_STATUS
                        ; Byte  1. (Data words following)
                        ; Byte  4. (Current Status Report Tag)
                        ; Word  0  (FAILURE   status)
```

6.27 REQUEST_VERSION_NUMBER

The REQUEST_VERSION_NUMBER instruction reports the version number and driver of PRO/GIDIS.

Opcode: 71 **Length:** 0

Format: REQUEST_VERSION_NUMBER

Status: SUCCESS

The report consists of 3 words:

Report header, driver, version

where

driver	Is 21 for Video GIDIS,
	22 for Plotter GIDIS,
	23 for Sixel GIDIS,
	24 for File GIDIS,
	25 for Palette GIDIS.
version	Is the version number of GIDIS.

Notes:

- For P/OS V2.0, the version number of GIDIS is 21.
- For P/OS V2.0A, the version number of GIDIS is 29.
- For P/OS V3.0, the version numer of GIDIS is 32.

Example:

```
        .BYTE   0.,71.  ;Length=0,
                        ;opcode for REQUEST_VERSION_NUMBER
                        ;byte 2. data words following
                        ;byte 7. VERSION_NUMBER_REPORT tag
                        ;word 21. device code
                        ;word 25. version number
```

6.28 SCROLL_CLIPPING_REGION

The SCROLL_CLIPPING_REGION instruction moves data within the clipping region. The vacated display area is set to the current secondary color.

Opcode: 52 **Length:** 2

Format: SCROLL_CLIPPING_REGION dx, dy

dx	The distance to move the data horizontally. If dx is positive, the data is shifted right to left; if negative, the data is shifted left to right.
dy	The distance to move the data vertically. If dy is positive, the data is shifted toward the top of the screen; if negative, the data is shifted toward the bottom of the screen.

Status: SUCCESS

Notes:

- The instruction applies to Video GIDIS only.
- SCROLL_CLIPPING_REGION is affected by the current plane mask. Planes not selected are not scrolled or otherwise changed.
- For speed, hardware assist is used when possible. When the clipping rectangle is all of IDS, a vertical scroll scrolls the entire width of the screen.
- When a software scroll is done, screen images appear to move around rather than scroll.
- The data scrolled out is not saved. You cannot scroll out a portion of an image and then scroll it back in. Solid secondary color always scrolls in.
- After this instruction, shaded areas within the clipping region will not necessarily be aligned with shaded areas outside the clipping region.

Example:

```
        .BYTE   2.,52.  ;Length=0,
                        ;opcode for SCROLL_CLIPPING_REGION
        .WORD   -100.   ;dx
        .WORD   0.      ;dy
                        ;Slides data to the right 100 units
```

Example:

```
        .BYTE   2.,52.
        .WORD   0.      ;Scroll data down
        .WORD   -15.    ;15 units
```

Example:

```
        .BYTE   2.,52.
        .WORD   -30     ;Scrolls data in the clipping region
        .WORD   +30     ;30 units left and 30 units up.
```

6.29 SET_ALPHABET

SET_ALPHABET sets the current alphabet to the specified alphabet. Except as noted below, all alphabet-related operations act on the currently selected apphabet.

Opcode: 38 **Length:** 1

Format: SET_ALPHABET alphabet

alphabet	Is an integer value in the range 0 to 15. It identifies the alphabet to make current.

Status: SUCCESS if the alphabet number is valid (from 0 to 15); otherwise, FAILURE.

Notes:

- A GIDIS alphabet number is somewhat like a character set. Alphabet 0 is the DEC Multinational Character Set. Alphabets 1 through 15 are user alphabets.
- The first time you select a nonzero alphabet number, there is no font for the alphabet. You get a font in one of two ways: the LOAD_BY_NAME instruction or the CREATE_ALPHABET instruction.
- SET_OUTPUT_CURSOR and SET_AREA_TEXTURE are the only alphabet related operations that do not act on the current alphabet.
- No drawing is done by the SET_ALPHABET instruction.

Example:

```
.BYTE   1.,38.  ;Length=1, opcode for
                ;SET_ALPHABET
.WORD   2.      ;Selects alphabet #2 as current
                ;alphabet
```

6.30 SET_AREA_CELL_SIZE

SET_AREA_CELL_SIZE clips the current area texture cell.

Opcode: 69 **Length:** 2

Format: SET_AREA_CELL_SIZE width, height

width	The width of the area cell in hardware pixels. The width value must be in the range 1 to 16.
height	The height of the area cell in hardware pixels. The height value must be in the range 1 to 16.

Status: SUCCESS if both width and height are in the range 1 to 16; otherwise, FAILURE.

Notes:

- If the area cell width is greater than the specified width, GIDIS removes columns from the right side of the area texture cell.
- If the area cell height is greater than the specified height, GIDIS removes columns from the bottom of the area texture cell.
- The SET_AREA_TEXTURE and SET_AREA_TEXTURE_SIZE instructions set the area cell size from that of the character cell, overriding any previous SET_AREA_CELL_SIZE specification.
- No drawing is done by the SET_AREA_CELL_SIZE instruction.

Device Note:

- Plotter GIDIS ignores this instruction.

Example:

```
        .BYTE   2.,14.  ;Length=2, opcode for SET_AREA_TEXTURE
        .WORD   2.      ;Use character 23 from alphabet 2,
        .WORD   23      ;which is 8 x 10
                        ;Area Cell Size is now 8 by 10
        .BYTE   2.,69.  ;Length=2, opcode for SET_AREA_CELL_SIZE
        .WORD   9.      ;area cell width = 9 hardware pixels
        .WORD   9.      ;area cell heighth = 9 hardware pixels
                        ;Area cell size is now 9 by 9, padded on
                        ;the right, with one column of OFF's, and
                        ;with the bottom row of the character
                        ;cell removed
```

6.31 SET_AREA_TEXTURE

SET_AREA_TEXTURE specifies the rectangular bit pattern used to fill subsequent filled figures. Area texture is specified as a character in an alphabet.

Opcode: 14 **Length:** 2

Format: SET_AREA_TEXTURE alphabet, char-index

alphabet	The number of the alphabet containing the texture character. It can be the DEC Multinational character set (alphabet 0), a user-defined alphabet, or the special texture alphabet -1.
char-index	The index of the character to use.

Status: SUCCESS if the specified alphabet number is valid; otherwise, FAILURE.

Notes

- The character identified by SET_AREA_TEXTURE is copied to an internal GIDIS storage area. Deleting or changing the font does not affect the current area texture. Only another SET_AREA_TEXTURE or SET_AREA_TEXTURE_SIZE can change the current area texture.

- Alphabet -1 char-index 0 asks GIDIS to derive the area texture cell from the current line texture. The pattern is drawn vertically and replicated horizontally. The area texture height is taken from the line texture size, not from the area texture width and height.

- Solid fill is more efficient than patterned fill. To generate solid fill most efficiently, select alphabet -1, character 0 while line texture is set to solid.

- Filling is most efficient when area texture width is 8 or 16.

- If the selected alphabet is associated with a family ID (See LOAD_BY_NAME(2)), SET_AREA_TEXTURE chooses a font for you using the current area texture size. This option gives you a way of making GIDIS generate more consistent patterns across devices of differing resolutions.

- The current font of the specified alphabet should not have width or height greater than 16 pixels. If width is greater than 16, only the leftmost 8 bits of the selected glyph are used. If height is greater than 16, only the topmost 16 lines of the glyph are used. If the alphabet is associated with a family ID, the selected font is guaranteed to be less than or equal to 16 x 16, if the current family contains such a font.

- The bit pattern specified by this instruction always appears upright; there is no way to rotate the pattern.

- Area textures are self-aligning. When two adjacent or overlapping areas are filled, no seams show.

- Complement and complement-negate writing modes can give unexpected results when filled figure areas overlap or abut.

- Pixel size is not used when filling areas.

- No drawing is done by the SET_AREA_TEXTURE instruction.

Device Note:

- Plotter GIDIS processes SET_AREA_TEXTURE differently. See Appendix E for a description of how Plotter GIDIS handles this instruction.

Example:

```
        .BYTE   2.,14   ;length=2., opcode for SET_AREA_TEXTURE
        .WORD   8.      ;User-defined alphabet 2
        .WORD   23.     ;23rd character
```

Example:

```
        .BYTE   3.,17   ;length=3.,opcode for SET_LINE_TEXTURE
        .WORD   any     ;length of pattern
        .WORD   -1      ;solid
        .WORD   any     ;size of one repetition of pattern
        .BYTE   2.,14   ;length=2.,opcode for SET_AREA_TEXTURE
        .WORD   -1      ;alphabet -1 is derived from line texture
        .WORD   0       ;character 0
                        ;sets up solid area fill
```

6.32 SET_AREA_TEXTURE_SIZE

SET_AREA_TEXTURE_SIZE specifies the desired size of the area texture cell.

Opcode: 3 **Length:** 2

Format: SET_AREA_TEXTURE_SIZE width, height

width	Specifies the width of one repetition of the area texture cell.
height	Specifies the height of one repetition of the area texture cell.

Status: SUCCESS if width and height are greater than zero; otherwise, FAILURE.

Notes:

- If the glyph you select (perhaps with GIDIS's help as described in the next note) for the area texture cell is smaller than the specified width and height, GIDIS scales the selected glyph to the size you specified. However, scaling is restricted to an integral multiple of the texture cell. GIDIS uses the largest multiple that is not larger than the size specified. If the glyph you select is larger than the specified width and height, GIDIS uses the selected glyph as is.

- If the selected alphabet is associated with a family ID (see LOAD_BY_NAME(2)), SET_AREA_TEXTURE_SIZE chooses a font for you using the current area texture. This option gives you a way of making GIDIS generate more consistent patterns across devices of differing resolutions.

- No drawing is done by the SET_AREA_TEXTURE_SIZE instruction.

Device Note:

- Plotter GIDIS ignores this instruction.

Example:

```
        .BYTE   2.,3    ;length=2,
                        ;opcode for SET_AREA_TEXTURE_SIZE
        .WORD   12.     ;Area texture width = 12 units
        .WORD   8.      ;Area texture height = 8 units
```

6.33 SET_CELL_DISPLAY_SIZE

SET_CELL_DISPLAY_SIZE defines the size of a character's display cell, the rectangular area of the view surface modified when a character is drawn.

Opcode: 40 **Length:** 2

Format: SET_CELL_DISPLAY_SIZE width, height

width	Is the width of the display cell.
height	Is the height of the display cell.

Status: SUCCESS

Notes:

- The origin of the display cell is always the upper left corner of the cell and is aligned with the unit cell at that corner.

- Normally display cell size and unit cell size are set the same. One reason to make display cell width wider than unit cell size is to space characters further apart. (See implicit movement in SET_CELL_MOVEMENT_MODE.) In Replace Writing mode this approach can be preferable to using SET_CELL_EXPLICIT_MOVEMENT, because there will not be gaps between the cells.

- If the unit cell is smaller than the display cell, all of the character is drawn and the right and bottom portions of the display cell are treated as if the character pattern specified OFF.

- If the unit cell is larger than the display cell, the bottom and right portions of the character are clipped to the display cell size. In other words, the character is truncated.

  Figure 6-5 shows what happens when the unit cell and display cell are not the same size.

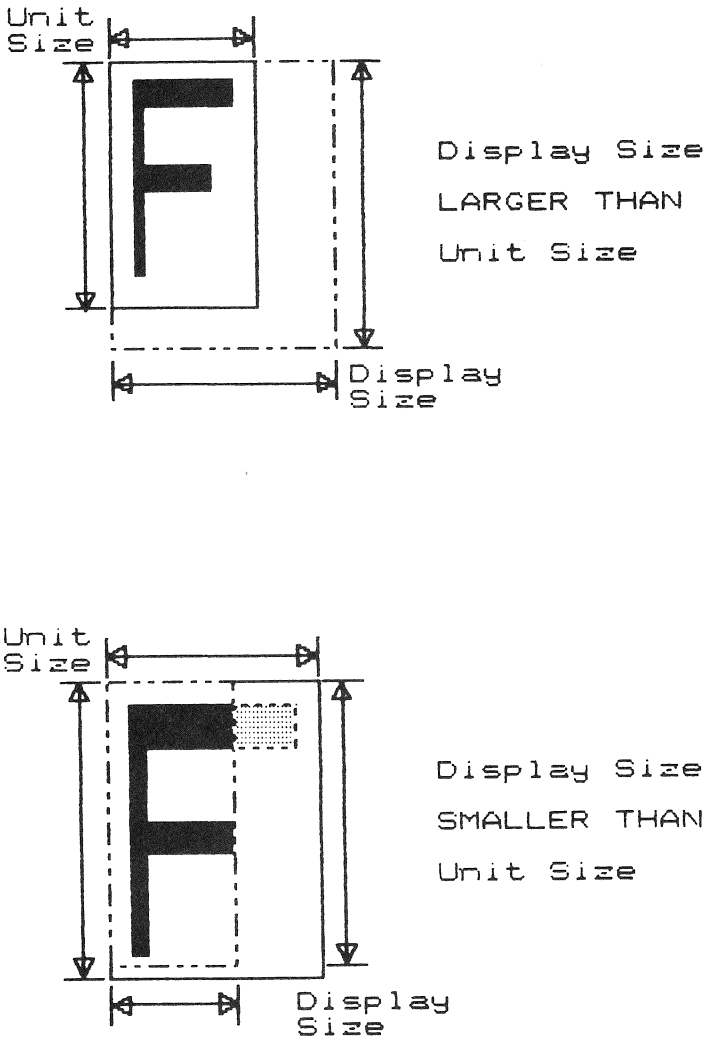


Figure 6-5: Character Unit Cell and Display Cell

- Negative values in width or height produce a mirroring in X and Y, respectively. This mirroring always occurs about the origin (the upper-left corner of the cell). Implicit movement always goes across the display cell; consequently, implicit movement for a display cell mirrored in X is in the opposite direction from the rotation angle.

- The smallest actual width or height is 1 hardware pixel.
- Display cell size, except for mirroring, is ignored for proportionally spaced fonts.
- No drawing is done by the SET_CELL_DISPLAY_SIZE instruction.

Example:

```
.BYTE   2.,40.  ;Length=2,
                ;opcode for SET_CELL_DISPLAY_SIZE
.WORD   12.     ;Width
.WORD   28.     ;Height
```

6.34 SET_CELL_EXPLICIT_MOVEMENT

SET_CELL_EXPLICIT_MOVEMENT specifies a distance to move the current position after a character is drawn.

Opcode: 41 **Length:** 2

Format: SET_CELL_EXPLICIT_MOVEMENT dx, dy

dx	Specifies the horizontal distance to move the current position.
dy	Specifies the vertical distance to move the current position.

Status: SUCCESS

Notes:

- Dx and dy define the total movement when implicit movement is OFF (see SET_CELL_MOVEMENT_MODE.)
- Explicit cell movement is not affected by SET_CELL_ROTATION or SET_CELL_OBLIQUE instructions.
- For left-to-right text, you normally use either implicit movement or explicit movement as follows:

Implicit Movement

```
        .BYTE    1.,42    ;Length=1,opcode for
                          ;SET_CELL_MOVEMENT_MODE
        .WORD    2        ;flag for implicit movement
        .BYTE    2.,41    ;length=2,opcode for
                          ;SET_CELL_EXPLICIT_MOVEMENT
        .WORD    0        ;dx
        .WORD    0        ;dy
```

Explicit Movement

```
        .BYTE    1.,42    ;length=1,opcode for
                          ;SET_CELL_MOVEMENT_MODE
        .WORD    0(or 1)  ;turns off implicit cell movement
        .BYTE    2.,41    ;length=2,opcode for
                          ;SET_CELL_EXPLICIT_MOVEMENT
        .WORD  width          ;as set by SET_CELL_DISPLAY_SIZE
        .WORD    0        ;
```

- No drawing is done by the SET_CELL_EXPLICIT_MOVEMENT instruction.

Example:

```
        .BYTE   2.,41.  ;Length=2,opcode for
                        ;SET_CELL_EXPLICIT_MOVEMENT
        .WORD   12.     ;dx
        .WORD   0.      ;dy
```

6.35 SET_CELL_MOVEMENT_MODE

SET_CELL_MOVEMENT_MODE specifies the manner in which the current position moves after a character is drawn.

Opcode: 42 **Length:** 1

Format: SET_CELL_MOVEMENT_MODE flag

flag	Specifies one of the following movement modes as shown in Table 6-5 below.

Table 6-5: SET_CELL_MOVEMENT_MODE Flag Values

Movement Mode	Value
Explicit cell movement, local symmetry	0
Explicit cell movement, global symmetry	1
Explicit and implicit movement, local symmetry	2
Explicit and implicit movement, global symmetry	3
Reserved	4-15

Status: SUCCESS if Flag is 0 to 3; otherwise, FAILURE.

Notes:

- Explicit cell movement is set by SET_CELL_EXPLICIT_MOVEMENT.
- Implicit movement means that the current position moves a distance equal to the display cell width. Movement is along the baseline of the angle of rotation. Thus, left-to-right text (aligned along a 0 degree angle) has implicit movement of dx = display cell width and dy = 0. Each successive

character is one display cell width to the right. Upwards perpendicular text (text aligned along a 90 degree angle), has implicit movement of dx = 0 and dy = -display cell width. Each successive character is one display cell width towards the top of the view surface.

- When using local symmetry, the current position after a DRAW_CHARACTERS instruction could be different from that calculated by your program. It is suggested that any series of DRAW_CHARACTERS instructions be followed by a SET_POSITION instruction or a REQUEST_POSITION instruction, unless you do not care exactly where the string ends.

- When using global symmetry, the final current position is exactly the value that would be calculated by your program. However, character spacing may not always be even due to round-off errors.

- For proportionally spaced fonts, you should normally specify implicit motion.

- If the current font is proportionally spaced, global symmetry is ignored.

- No drawing is done by the SET_CELL_MOVEMENT_MODE instruction.

Example:

```
.BYTE   1.,42.  ;Length=1,
                ;opcode for SET_CELL_MOVEMENT_MODE
.WORD   0.      ;Set explicit movement, local symmetry
```

6.36 SET_CELL_OBLIQUE

Normally a character cell is a rectangle. An oblique character cell is a parallelogram. SET_CELL_OBLIQUE specifies how much to slant the rectangle. The top line of the display cell remains stationary, while the bottom of the cell is moved either forward or backward.

Opcode: 65 **Length:** 1

Format: SET_CELL_OBLIQUE angle

angle	The requested angle in degrees. A positive angle value indicates a backward slant (move bottom of cell forwards); a negative angle indicates a forward slant (move bottom of cell backwards)

Status: SUCCESS

Notes:

- If the specified angle is greater than 60 (or less than -60), GIDIS uses 60.
- The shape of the parallelogram is not affected by cell rotation.
- If x of the current position is at the left edge of the clipping rectangle, the bottom left of a forward-slanted character will be clipped. This is because a forward slant is achieved by moving the bottom of the cell backwards, rather than by moving the top of the cell forwards.
- No drawing is done by the SET_CELL_OBLIQUE instruction.

Example:

```
        .BYTE   1.,65.  ;Length=1, opcode for SET_CELL_OBLIQUE
        .WORD   -23.    ;Approximate italics (slant right
                        ;23 degrees)
```

Example:

```
        .BYTE   1.,65.  ;Length=1, opcode for SET_CELL_OBLIQUE
        .WORD   23.     ;Approximate back-slant (slant left
                        ;23 degrees
```

Figure 6-6 shows the two display cells that result from the above examples.

Figure 6-6: Italic and Back-Slanted Display Cells

6.37 SET_CELL_RENDITION

SET_CELL_RENDITION controls character rendition. The rendition options defined for the Professional are: back-slant, italics, bold, and proportionally spaced text.

Opcode: 43 **Length:** 1

Format: SET_CELL_RENDITION flags

flags — A word that specifies zero or more of the cell rendition options. A rendition is represented by the value of set bits as shown in Table 6-6. A 0 value establishes normal rendition: no slant, not bold, not proportional.

Table 6-6: SET_CELL_RENDITION Flags

Cell Rendition	Bit	Value
Back Slant	0	1
Italics	1	2
Bold	3	8
Proportionally spaced	4	16
Reserved	2, 5-15	

Status: SUCCESS

Notes:

- SET_CELL_RENDITION is not cumulative. A SET_CELL_RENDITION with an argument of 2 followed by another SET_CELL_RENDITION with an argument of 8 causes the character to be bold **not** slanted, rather than bold **and** slanted.
- SET_CELL_RENDITION with an argument of 3 (mask value B00011) selects italics.
- SET_CELL_RENDITION simulates a SET_CELL_OBLIQUE with an argument of 0, before processing the mask-value. This cancels the effect of earlier SET_CELL_OBLIQUE instructions.
- If possible, GIDIS satisfies a request for bold or italics by using a font with that attribute. Otherwise, it algorithmically creates the desired rendition.
- If font width size is less than or equal to 8 pixels, algorithmic bolding has no visible effect.
- Algorithmic italics is equivalent to SET_CELL_OBLIQUE -23.
- Algorithmic backslant is equivalent to SET_CELL_OBLIQUE 23.
- There is no algorithmic fallback for proportionally spaced text. The proportional bit is used only to request a proportional font from a font family. If no appropriate font is found, the argument is ignored. See LOAD_BY_NAME(2).
- For proportionally spaced fonts, you should normally specify implicit motion.
- If the current font is proportionally spaced, global symmetry is ignored.

Example:

```
.BYTE   1.,43.  ;Length=1, opcode for SET_CELL_RENDITION
.WORD   2.      ;Requests italics rendition
```

6.38 SET_CELL_ROTATION

SET_CELL_ROTATION defines the angle of rotation with which subsequent characters are drawn. The character is rotated about the current position (upper left corner of the display cell) to the angle specified.

Opcode: 44 **Length:** 1

Format: SET_CELL_ROTATION angle

angle	The requested angle in degrees. A positive angle value indicates counter-clockwise from normal text. For example, 90 degree text is sideways, facing up. A negative angle indicates clockwise from normal text.

Status: SUCCESS

Notes:

- The simplest way to make a string of rotated characters follow a baseline is to use SET_CELL_EXPLICIT_MOVEMENT with arguments of [0,0] and set the implicit movement flag with the SET_CELL_MOVEMENT_MODE instruction.
- An angle of N-360 is equivalent to an angle of N.
- No drawing takes place when the SET_CELL_ROTATION instruction executes.

Example:

```
.BYTE   1.,44.   ;Length=1, opcode for SET_CELL_ROTATION
.WORD   -90.     ;Text to face down the screen
```

6.39 SET_CELL_UNIT_SIZE

SET_CELL_UNIT_SIZE specifies the size to draw subsequent character cells.

Opcode: 45 **Length:** 2

Format: SET_CELL_UNIT_SIZE width, height

width	Is the desired cell width. The width must be greater than zero.
height	Is the desired cell height. The height must be greater than zero.

Status: SUCCESS if width and height are greater than zero; otherwise, FAILURE.

Notes:

- If the current alphabet is associated with a font family ID (See LOAD_BY_NAME(2)), GIDIS tries to find a font in the family whose size matches the specified size. If the size request is between two available fonts, GIDIS generally selects the smaller of the two.
- If the current alphabet has a CREATE_ALPHABET font or a LOAD_BY_NAME(1) font, GIDIS must use that font.
- If the available (or chosen) font does not match the specified size, GIDIS scales it. Width and height are scaled independently.
- GIDIS may not be able to scale the font exactly to the specified size. However, unit cell size will not be exceeded unless the specified size is less than half the size of the font being scaled.
- The width of a proportionally spaced font can only be scaled to an integral multiple of itself.
- The requested unit size does not change when the current alphabet changes, but the the font and/or scaling is recalculated in order to obtain the best match.
- The unit cell and the display cell are always aligned at their upper-left corners.

- Normally when you change unit cell size, you would also make an analogous change to display cell size.
- No drawing is done by the SET_CELL_UNIT_SIZE instruction.

Device Notes:

- Plotter GIDIS always sets unit cell size to display cell size.
- Plotter GIDIS always sets unit cell size exactly.

Example:

```
        .BYTE   1.,45.  ;Length=1, opcode for SET_CELL_UNIT_SIZE
        .WORD   10.     ;Width
        .WORD   30.     ;Height
```

6.40 SET_COLOR_MAP_ENTRY

The SET_COLOR_MAP_ENTRY instruction sets the specified color map entry. All pixels that were previously drawn using that color map entry are immediately affected.

Opcode: 16 **Length:** 6

Format: SET_COLOR_MAP_ENTRY map, index, red, green, blue, mono

map	an integer representing a specific color map. For the Professional, this value must be 0.
index	an integer from 0 to (color map size -1), to identify the color map entry.
red	an integer in the range of 0 to 65535, representing the intensity of red.
green	an integer in the range of 0 to 65535, representing the intensity of green.
blue	an integer in the range of 0 to 65535, representing the intensity of blue.
mono	an integer in the range of 0 to 65535, representing the intensity of monochrome.

Status: SUCCESS if map = 0 and index is in range; otherwise, FAILURE.

Notes:

- Table 6-7 shows color intensities in various formats: octal fraction, octal integer, unsigned decimal integer and signed decimal integer. Making an intensity value larger linearly increases intensity. For example, specifying 32768 sets a color to half its maximum intensity, while specifying 65535 sets a color to its maximum intensity.

Table 6-7: Sample Color Map Values

Octal Fraction	Octal Integer	Unsigned Decimal Integer	Signed Decimal Integer
0.0	0	0	0
0.1	20000	8192	8192
0.2	40000	16384	16384
0.3	60000	24576	24576
0.4	100000	32768	-32768
0.5	120000	40960	-24576
0.6	140000	49152	-16384
0.7	160000	57344	-8192
Max	177777	65535	-1

Device Notes:

- Video GIDIS ignores this instruction unless the system has an Extended Bitmap Option (EBO).
- On a Video with an EBO, there are 8 color map entries. On Palette, there are 16 color map entries.
- On Professional 350 Video, there are 8 intensity levels available for mono, red, and green; and 4 for blue.
- On Professional 380 Video and Palette, there are 16 intensity levels available for mono, red, green, and blue.

- If any in-between value is specified, the next lower value is used. However, how "in between" values are treated is device specific. For example, the PRO 380 treats 0.16 (octal) for red as 0.15, while the PRO 350 treats 0.16 (octal) as 0.1.

- Sixel GIDIS and Plotter GIDIS ignore this instruction.

Example:

```
.BYTE   6.,16.  ;length=6,opcode for SET_COLOR_MAP_ENTRY
.WORD   0.      ;PRO's bitmap (value must be 0)
.WORD   1.      ;Color index 1
.WORD   49152   ;Red is three-quarters
.WORD   40960.  ;Green is five-eighths
.WORD   0       ;Blue is zero
.WORD   32768.  ;Monochrome is one-half
                ;This makes dark yellow on a color
                ;system.
```

6.41 SET_GIDIS_OUTPUT_SPACE

SET_GIDIS_OUTPUT_SPACE specifies the coordinate units and shape of a window you define in GOS. Simultaneously, it sets the output clipping rectangle to coincide with the window. This instruction also resets GIDIS attributes as shown in Table 6-8.

Opcode: 9 **Length:** 4

Format: SET_GIDIS_OUTPUT_SPACE ulx, uly, width, height

ulx	Is the x value to assign to the leftmost point in your window.
uly	Is the y value to assign to the topmost point in your window. In other words, [ulx, uly] is the origin of the window.
width	Specifies the number of x units in your window (See second note below.)
height	Specifies the number of y units in your window (See second note below.)

Status: SUCCESS if width and height are greater than zero; otherwise, FAILURE.

Notes:

- When drawing a picture with GIDIS, you normally just use a SET_OUTPUT_IDS instruction. This sets your viewport and clipping rectangle to the entire view surface, and makes GOS units and IDS units the same. For example, if the view surface width in IDS is 960, then the view surface width in GOS is also 960. You only need to use SET_GIDIS_OUTPUT_SPACE if you do not want to draw on the entire view surface or if you want to draw only a portion of a picture on the view surface. (See the third note.)

- If the window shape is not the same as the viewport shape, then space is left unused to the right or bottom of the picture. Figure 6-7 shows how a window with an extent of 1000 x 1000 maps to a viewport with an extent of 1500 x 1000. Note how GIDIS begins at the upper left-hand corner and fills as much of the viewport as possible. In this case the vertical extents match, so the picture extends to the bottom of the viewport. Since the horizontal extents do not match, GIDIS leaves space on the right.

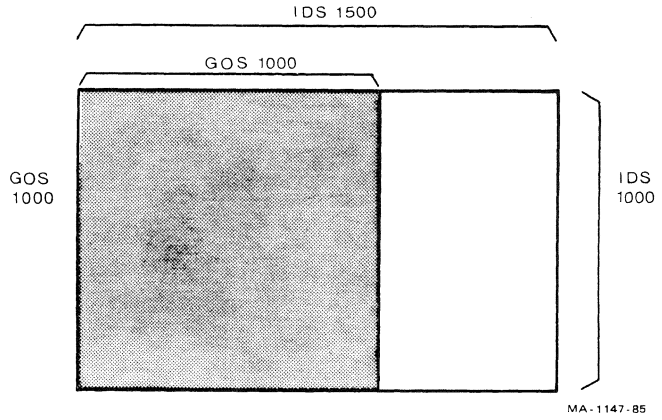


Figure 6-7: Mapping of GOS to a Different Shaped Viewport

- The SET_GIDIS_OUTPUT_SPACE instruction makes it easy to display a selected rectangle from a larger picture. Choose the portion of the picture you want. Use its origin, width, and height as arguments to SET_GIDIS_OUTPUT_SPACE, and then draw the whole picture. GIDIS will fill your viewport with the desired picture section and clip away the rest. The effect is that of enlarging a portion of your picture, while maintaining all existing proportions. (However, if the GIDIS instructions that comprise the whole picture include a SET_OUTPUT_GOS or SET_OUTPUT_IDS, you will not achieve the desired result.) Figure 6-8 shows how a selected portion of a GIDIS picture maps to a viewport with an extent in IDS of 500 x 500. The area to be drawn is identified by the following arguments:

```
ulx = 300
uly = 100
width = 100
height = 100
```

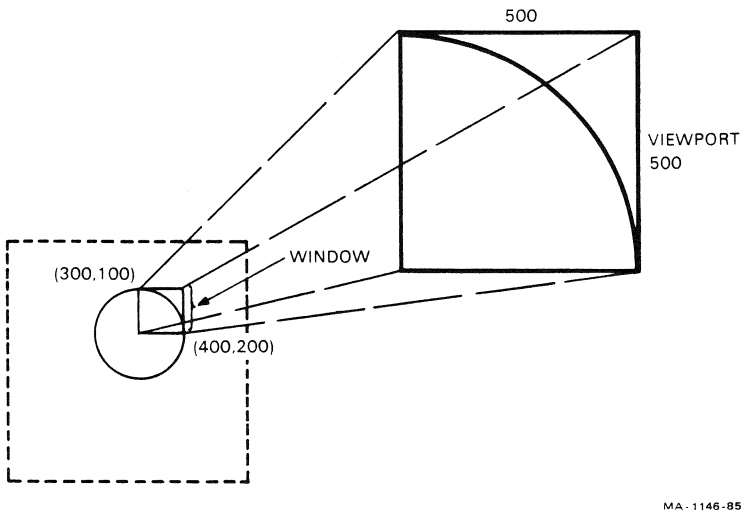


Figure 6-8: Mapping a Portion of a Picture to a Viewport

- It is recommended that ulx, uly, (ulx + width), and (uly + height) never be set larger than 16384 (2 to the 14th power). This will allow sufficient off-screen address space for accurate clipping.

- No drawing is done by the SET_GIDIS_OUTPUT_SPACE instruction.

- Table 6-8 lists all of the GIDIS attributes modified by the SET_GIDIS_OUTPUT_SPACE instruction. Standard text size is the number of GOS units needed to match hardware character size to the size set by INITIALIZE -1.

Table 6-8: GIDIS Attributes Affected by SET_GIDIS_OUTPUT_SPACE

Attribute	Value
GIDIS output space	as specified
clipping region	same as GOS
current position x	0
current position y	0
line texture size	N/A
area texture width	12
area texture height	25
logical pixel x offset	0
logical pixel y offset	0
logical pixel width	0
logical pixel height	0
cell movement mode flag	2, (implicit)
cell explicit movement dx	0
cell explicit movement dy	0
cell display size width	12
cell display size height	25
cell unit size width	12
cell unit size height	25

Example:

```
                    ;assume Video GIDIS
        .BYTE   2.,12.  ;length=2,opcode for SET_OUTPUT_IDS
        .WORD   960.    ;width
        .WORD   600.    ;height (upper left corner is [0,0] and
                        ;  lower right corner is [959,599])

        .BYTE   4.,13.  ;length=4,opcode for SET_OUTPUT_VIEWPORT
        .WORD   0.      ;
        .WORD   0.      ;Sets the viewport to the left half
        .WORD   480.    ;   of the screen
        .WORD   600.    ;
```

```
.BYTE   4.,9.   ;length=4,opcode SET_GIDIS_OUTPUT_SPACE
.WORD   0.      ;Sets GOS to 0 to 2399 in X
.WORD   0       ;  and 0 to 2999 in Y  (all within the
.WORD   2400.   ;  left half of the screen).
.WORD   3000.   ;Because 480/600 = 2400/3000, there is
                ;  no wasted space at the bottom or
                ;  right of the viewport.
```

6.42 SET_LINE_TEXTURE

SET_LINE_TEXTURE defines the line texture, a bit pattern that is scaled and repeated in drawing straight lines and arcs.

Opcode: 17 **Length:** 3

Format: SET_LINE_TEXTURE patlen, pattern, size

patlen	Is the length (in bits) of the specified pattern. It must be in the range 1 to 16.
pattern	Is the bit pattern to use. PRO/GIDIS begins the pattern by using the low-order (rightmost) bit (bit 0) first.
size	Specifies the length of pattern repetition in GOS units. It must be greater than zero.

Status: SUCCESS if patlen is in the range 1 to 16, and if size is greater than 0; otherwise, FAILURE.

Notes:

- The size argument in this instruction is handled much like size in the SET_CELL_UNIT_SIZE instruction. However, scaling is limited to integral multiples of the pattern.
- Drawing with a solid pattern (that is, pattern = -1) is quite a bit more efficient than drawing with a nonsolid pattern.
- When specifying a nonsolid pattern, a highly multiplied pattern is best. For example, patlen = 2 and pattern = 1 are more efficient than patlen = 16 and pattern = 255 for drawing a dashed line.
- Pixel size does not change how often the pattern repeats, although the appearance of the line does change somewhat.
- The pattern is rotated as lines are drawn, so that the pattern is preserved around corners and bends.
- Conversely, the only way to force the pattern to bit 0 is to issue another SET_LINE_TEXTURE instruction.
- Except for Plotter GIDIS, the size given is used only for horizontal and vertical lines. Diagonal lines have a size that is larger by as much as a factor of the square root of 2 (1.414...).

- No drawing is done by the SET_LINE_TEXTURE instruction.

Device Notes:

- For Plotter GIDIS, the specified pattern is mapped to the closest pattern provided by the plotter hardware. The patterns provided are solid, dashes, long dashes, long dash short dash, long short short, and dots. The size of the line pattern is set to the value specified in the command, with a minimum of about .125 inches. The pattern is rotated rather than projected when a diagonal line is drawn.

- Plotter GIDIS maintains the correct size regardless of the direction of the lines.

Example:

```
.BYTE   3.,17   ;length=3., opcode for SET_LINE_TEXTURE
.WORD   8.      ;patlen
.WORD   ^B10001111 ;ON, ON, ON, ON, OFF, OFF, OFF, ON
                ;Bits are used in order low to high
.WORD   100.    ;Size of one repetition of pattern in
                ;GIDIS output space
                ;makes subsequent lines dashed
```

6.43 SET_OUTPUT_BITMAP

SET_OUTPUT_BITMAP tells GIDIS which page of the 380 video bitmap to make current. All drawing executes on the current bitmap.

Opcode: 145 **Length:** 2

Format: SET_OUTPUT_BITMAP bitmap-no, dis-flag

bitmap-no	Specifies which bitmap to make current. In low resolution mode, the value can be 1 to 4. In high resolution mode, the value can be 1 to 2.
dis-flag	Controls whether the current bitmap is to be displayed. If flag is set, the current bitmap is displayed. If flag is not set, the current bitmap is not displayed.

Status: SUCCESS

Notes:

- Each bitmap is a complete environment with its own color map.
- SET_OUTPUT_BITMAP does not alter any other GIDIS attributes. It only affects the current bitmap.
- Do a SET_OUTPUT_BITMAP with a dis-flag of 0, if you want to continue looking at the current picture on the screen while GIDIS draws the next picture. This is a useful feature in a slide show application, for instance.
- SET_OUTPUT_BITMAP is available on the Professional 380 only.

Device Notes:

- You may scroll the displayed bitmap, provided you do not write to that bitmap while it is not the displayed bitmap.
- Always go back to bitmap 1 before using text mode. You can do this in several ways:
  - Use SET_OUTPUT_BITMAP with a bitmap-no argument of 1
  - Use the DCL command CLEAR
  - Use a RIS escape sequence

6.44 SET_OUTPUT_CLIPPING_REGION

SET_OUTPUT_CLIPPING_REGION specifies the output clipping rectangle. The clipping rectangle is the area on the view surface where PRO/GIDIS can draw.

Opcode: 4 **Length:** 4

Format: SET_OUTPUT_CLIPPING_REGION ulx, uly, dx, dy

ulx	Specifies the x coordinate of the left edge of the clipping region.
uly	Specifies the y coordinate of the top edge of the clipping region.
dx	Sets the rightmost x of the clipping rectangle to ulx + dx.
dy	Specifies the bottommost y of the clipping rectangle to uly + dy.

Status: SUCCESS if width and height are not negative; otherwise, FAILURE.

Notes:

- You cannot set the clipping region to an area larger than the device's Hardware Address Space (HAS). An attempt to do so reduces the clipping region to the available space.
- Clipping does not affect the setting of the current position. For example, if you draw a line that ends outside the clipping rectangle, the current position is set to the x and y you specified, even though only part of the line was drawn.
- Clipping applies to all drawing. For example, any part of a character outside of the clipping region is not drawn.
- Clipping does not affect drawing accuracy. In particular, if only part of an arc is inside the clipping region, that part is drawn correctly.

- Because the clipping rectangle includes the right and bottom borders,

```
SET_POSITION 100, 150
DRAW_LINES 500, 150
```

  draws pixels from [100,150] to [400,150] inclusive. Note that the current position is now [500,150].

- No drawing is done by the SET_OUTPUT_CLIPPING_REGION instruction.

Example:

```
.BYTE   4.,4.    ;length=4,
                 ;opcode for SET_OUTPUT_CLIPPING_REGION
.WORD   100.     ;Sets output clipping region to rectangle
.WORD   100.     ;with the upper left corner at 100,100
.WORD   400.     ;and the lower right corner at 500,200.
.WORD   100.
```

6.45 SET_OUTPUT_CURSOR

SET_OUTPUT_CURSOR selects the symbol to be used as the cursor and aligns it relative to the current position. The cursor is a visible indication of the current position.

Opcode: 5 **Length:** 6

Format: SET_OUTPUT_CURSOR alphabet, index, width, height, offset-x, offset-y

alphabet	Specifies the alphabet containing the character or the special cursor alphabet (-1).
index	Specifies the character or special cursor.
width	Specifies the width of the cursor. The width must be greater than or equal to zero.
height	Specifies the height of the cursor. The height must be greater than or equal to zero.
offset-x	Specifies distance from the left edge of the cursor to the current position (range is 0 to width).
offset-y	Specifies distance from the top edge of the cursor to the current position (range is 0 to height).

Status: SUCCESS if alphabet is -1 to 15, alphabet width is less than or equal to 16, alphabet height is less than or equal to 16, and the offsets are in range; otherwise, FAILURE.

Notes:

- Applies to Video GIDIS only.
- If the alphabet is not -1, the width and height are treated as a unit cell size; there is no equivalent of a display cell. When the specified character is scaled to width and height (using the rules described under SET_AREA_TEXTURE_SIZE, the x and y offsets are scaled by the same ratios.

- An alphabet code of -1 specifies that one of the following special built-in cursors should be used:

-1	No cursor
0	Implementation default (same as 1)
1	Tracking cross (small cross)
2	Crosshairs (full clipping rectangle width and height)
3	Block (solid rectangle)
	All other values are reserved.

  Width, height, and the offsets are ignored when the tracking cross or crosshairs are used.

- If the chosen cursor is neither a special cursor nor a character in alphabet 0, your program must define the character before executing SET_OUTPUT_CURSOR. Redefining the selected character after a SET_OUTPUT_CURSOR does not change the cursor's appearance. You must use another SET_OUTPUT_CURSOR to change the appearance of the cursor.

- When the SET_OUTPUT_CURSOR instruction executes, the appearance of the cursor changes immediately.

Device Note:

- SET_OUTPUT_CURSOR changes only the GIDIS cursor. However, turning the Terminal Subsystem's text cursor ON or OFF has the side effect of turning the Video GIDIS cursor ON or OFF.

Example:

```
.BYTE   6.,5.   ;length=6,opcode for SET_OUTPUT_CURSOR
.WORD   1.      ;Alphabet 1 (user-defined alphabet)
.WORD   2.      ;Character index value
                ;(Assume that Alphabet 1, character-index
                ;2, is defined as an arrow pointing
                ;straight upward
.WORD   30.     ;Width of 30
.WORD   30.     ;Height of 30
.WORD   15.     ;offset-x
.WORD   0.      ;offset-y
                ;Makes the arrow the new cursor and
                ;aligns it such that its tip is at the
                ;current position.
```

Example:

```
        .BYTE   6.,5.   ;length=6,opcode for SET_OUTPUT_CURSOR
        .WORD   -1.     ;PRO/GIDIS Cursor Alphabet
        .WORD   -1.     ;No cursor
        .WORD   0.      ;Width value of zero  (ignored)
        .WORD   0.      ;Height value of zero  (ignored)
        .WORD   3.      ;offset-x (ignored)
        .WORD   4.      ;offset-y (ignored)
                        ;turns the GIDIS cursor off
```

6.46 SET_OUTPUT_CURSOR_RENDITION

SET_OUTPUT_CURSOR_RENDITION controls cursor and rubber band rendition. The rendition options are blinking and continuous.

Opcode: 72 **Length:** 1

Format: SET_OUTPUT_CURSOR_RENDITION mask

mask	Is a word that specifies the rendition. The rendition is represented in the mask as a bit. If the 0 bit is set, the cursor or rubber band blinks; if not, it is continuous.

Status: SUCCESS

Notes:

- Applies to Video GIDIS only.
- Bits 1-15 of mask are reserved.
- Using a continuous cursor during picture drawing is very expensive.

Example:

```
        .BYTE   1.,72.  ;length=1, opcode for
                        ;SET_OUTPUT_CURSOR_RENDITION
        .WORD   0.      ;set to continuous mode
```

6.47 SET_OUTPUT_IDS

SET_OUTPUT_IDS specifies the width and height of Imposed Device Space (IDS). It also sets GIDIS Output Space, the clipping rectangle, and the viewport to be identical with IDS, and sets all GIDIS attributes as shown in Table 6-9.

Opcode: 12 **Length:** 2

Format: SET_OUTPUT_IDS width, height

width	Specifies the number of x units on your device
height	Specifies the number of y units on your device

Status: SUCCESS if width and height are greater than 0; otherwise, FAILURE.

Notes:

- The upper left corner of IDS is always [0,0]. The coordinates of the lower-right corner are [width -1, height -1].
- When the shape of IDS is not equal to the shape of the hardware address space, only the top left portion of the view surface is used. This mapping mirrors the mapping of GOS to a different shaped viewport. See note 2 under SET_GIDIS_OUTPUT_SPACE.
- It is recommended that width and height never be set larger than 16384 (2 to the 14th power). This will allow sufficient off-screen address space for accurate clipping.
- No drawing is done by the SET_OUTPUT_IDS instruction.
- Table 6-9 lists all of the GIDIS attributes affected by the SET_OUTPUT_IDS instruction.

Table 6-9: GIDIS Attributes Affected by SET_OUTPUT_IDS

Attribute	Value
IDS width	as specified
IDS height	as specified
viewport	same as IDS
GIDIS output space	same as IDS
clipping region	same as IDS
current position x	0
current position y	0
line texture size	N/A
area texture width	12
area texture height	25
logical pixel x offset	0
logical pixel y offset	0
logical pixel width	0
logical pixel height	0
cell movement mode flag	2 (implicit)
cell explicit movement dx	0
cell explicit movement dy	0
cell display size width	12
cell display size height	25
cell unit size width	12
cell unit size height	25

Example:

```
                        ;assume Video GIDIS
        .BYTE   2.,12.  ;length=2,opcode for SET_OUTPUT_IDS
        .WORD   960.    ;width
        .WORD   600.    ;height (upper left corner is [0,0] and
                        ;  lower right corner is [959,599])

        .BYTE   4.,13.  ;length=4,opcode for SET_OUTPUT_VIEWPORT
        .WORD   0.      ;
        .WORD   0.      ;Sets the viewport to the left half
        .WORD   480.    ;   of the screen
        .WORD   600.    ;

        .BYTE   4.,9.   ;length=4,opcode SET_GIDIS_OUTPUT_SPACE
        .WORD   0.      ;
        .WORD   0.      ;Sets GOS to 0-to-2399 in X
        .WORD   2400.   ;  and 0-to-2999 in Y  (all within the
        .WORD   3000.   ;      left half of the screen)
                        ;Because 480/600 = 2400/3000, there is
                        ;no wasted space at the bottom and
                        ;right of the viewport.
```

6.48 SET_OUTPUT_RUBBER_BAND

SET_OUTPUT_RUBBER_BAND specifies if a rubber band is to be generated. It also gives the origin of the rubber band.

Opcode: 53 **Length:** 3

Format: SET_OUTPUT_RUBBER_BAND type, origin-x, origin-y

type	Is the type of rubber band to use. (See table 6-10.)
origin-x	Is the x coordinate of the desired rubber band's origin.
origin-y	Is the y coordinate of the desired rubber band's origin.

Status: SUCCESS if the type is legal; otherwise, FAILURE.

Table 6-10: Types of Rubber Bands

Type Code	Rubber Band
-1	No rubber band
0	Default (same as -1)
1	Rubber band line
2	Rubber band rectangle

Notes:

- Applies to Video GIDIS only.
- The SET_OUTPUT_CURSOR_RENDITION instruction applies to rubber bands as well as cursors.

- The rubber band line is drawn from its origin to the current position.

- The rubber band rectangle is a rectangle with one corner at the rubber band origin and the opposite corner at the current position. The rectangle will degenerate to a line (or point) if one or both of the coordinates of the current position and rubber band origin are the same.

- Since both the cursor and the rubber band are drawn in complement mode, it may be preferable to turn the cursor OFF when a rubber band is ON.

Example:

```
        .BYTE   3.,53.  ;length=3., opcode for
                        ;SET_OUTPUT_RUBBER_BAND
        .WORD   1.      ;rubber band line
        .WORD   50.     ;its origin is [50,60]
        .WORD   60.

        .BYTE   2.,29.  ;length=1., opcode for SET_POSITION
        .WORD   100.    ;new current position
        .WORD   300.    ;is [100,300]

                        ;there will be a rubber band line from
                        ;[50,60] to [100,300].
```

6.49 SET_OUTPUT_VIEWPORT

SET_OUTPUT_VIEWPORT specifies the size and location of your viewport. Your viewport is the rectangle on the view surface to which your picture is mapped for display. SET_OUTPUT_VIEWPORT also sets the clipping rectangle to match the viewport.

Opcode: 13 **Length:** 4

Format: SET_OUTPUT_VIEWPORT ulx, uly, width, height

ulx	specifies the x coordinate of the left edge of the viewport, in IDS units
uly	specifies the y coordinate of the top edge of the viewport, in IDS units
width	specifies the width of the viewport, in IDS units
height	specifies the height of the viewport, in IDS units

Status: SUCCESS if width and height are greater than 0; otherwise, FAILURE.

Notes:

- Use this instruction when you want the drawing area to be smaller than the view surface.
- To copy a picture to another part of the view surface and/or change its size, you need only do a SET_OUTPUT_VIEWPORT and then redraw the picture. You need to do a SET_GIDIS_OUTPUT_SPACE as well only if you want to draw a different portion of the picture.
- Unlike SET_OUTPUT_IDS and SET_GIDIS_OUTPUT_SPACE, this instruction does not initialize any of the GIDIS attributes. However, it does alter them. For example, suppose cell unit width in GOS is 36 and you make your viewport half as wide. This makes every GOS unit half as wide. Thus if cell unit width had been 18 pixels, it is now 9 pixels. Cell unit width in GOS is still 36, but 36 GOS units is half as wide as before.

- A SET_OUTPUT_IDS simulates a SET_OUTPUT_VIEWPORT with arguments as follows:

```
ulx    = 0
uly    = 0
width  = width of IDS
height = height of IDS
```

- No drawing is done by the SET_OUTPUT_VIEWPORT instruction.

Example: See SET_GIDIS_OUTPUT_SPACE description.

6.50 SET_PIXEL_SIZE

SET_PIXEL_SIZE permits you to set the size of the logical pixel used for drawing straight lines and arcs. For large pixels, you also control where the pixel is aligned relative to the current position.

Opcode: 19 **Length:** 4

Format: SET_PIXEL_SIZE width, height, offset-x, offset-y

width	Specifies the width of the logical drawing pixel.
height	Specifies the height of the logical drawing pixel.
offset-x	Specifies distance from the left edge of the pixel to the current position.
offset-y	Specifies distance from the top edge of the pixel to the current position.

Status: SUCCESS if width and height are greater than or equal to zero, offset-x is greater than or equal to zero and not greater than width, and offset-y is greater than or equal to zero and not greater than height; otherwise, FAILURE.

Notes:

- The drawing pixel is always a rectangle orthogonal to the X and Y axes.
- Changing pixel size does not change the size of GOS units. It just tell GIDIS the size and alignment of the rectangle to draw at each point along the line. Thus patterned lines have less "off space" when pixel size is large.
- Default pixel size is device dependent. It is between 1/50 and 1/100 of an inch. If possible, one hardware pixel is used.
- A size value that maps to a size smaller than a hardware pixel is set to the hardware pixel size. However, width = 0 and height = 0 sets the logical drawing pixel to the default pixel size.

- Because the pixel is a rectangle, a diagonal line is thicker than a horizontal or vertical line.
- When pixel size is not 1 x 1, complement writing mode can produce unexpected results.
- No drawing is done when the SET_PIXEL_SIZE function executes.

Device Notes:

- On Plotter GIDIS, SET_PIXEL_SIZE sets line width to (width + height)/2.
- On Plotter GIDIS, one hardware pixel is the size of the pen.
- For purposes of drawing thick lines on a plotter, a hardware pixel is treated as 1/75 of an inch. However a double line is not drawn until line width is greater than 1/30 of an inch. This is to accommodate the fact that a .7mm pen is almost this thick.

Example:

```
.BYTE   4.,19.  ;length=4,opcode for SET_PIXEL_SIZE
.WORD   6.      ;width in GIDIS output space units
.WORD   6.      ;Height in GIDIS output space units
.WORD   3.      ;Centers the current position
                ;horizontally
.WORD   3.      ;Centers the current position vertically
```

6.51 SET_PLANE_MASK

SET_PLANE_MASK performs a Boolean AND operation on the plane-mask and the current color index and sets pixels using the resultant index. For example, if the current color index is 5 and the plane-mask is 3, a color index 1 (=3 AND 5) is actually used.

Opcode: 20 **Length:** 1

Format: SET_PLANE_MASK plane-mask

plane-mask	Is a bit mask representing a combination of planes. A set bit indicates an accessible plane.

Status: SUCCESS

Notes:

- Use a mask of -1 to ensure that all planes are accessible.
- No drawing is done by the SET_PLANE_MASK instruction.

Device Notes:

- When used with an EBO, the text portion of the Terminal Subsystem uses plane 3 for text. When not used with an EBO, it uses plane 1 for text.
- The various GIDIS devices have different numbers of planes:
  - Professional Video with the EBO has 3 planes.
  - Palette has 4 planes.
  - Plotter GIDIS has 3 planes.
  - All other devices have 1 plane.
- In Video GIDIS the color map can be used in combination with the plane mask to prepare separate images in separate planes for switching back and forth quickly. For example:

```
        .               ;Set all color map entries to dark
        .               ;and clear bitmap
        .

    .BYTE   1.,20.  ;length=1,opcode for SET_PLANE_MASK
```

```
.WORD   1.          ;plane 1
  .
  .                 ;draw image A in plane 1
  .
  .                 ;Set color map entry 1 to desired color
  .                 ;Image A appears
  .

.BYTE   1.,20.      ;length=1,opcode for SET_PLANE_MASK
.WORD   2.          ;plane 2
  .
  .                 ;draw image B in plane 2
  .
  .                 ;Set color map entry 1 to dark
  .                 ;Image A disappears
  .
  .                 ;Set color map entry  2 to desired color
  .                 ;Image B appears
  .
```

However long it took to draw Image B, it will appear all at once. You can continue flipping images A and B very quickly. In other words, you can draw B while A is being viewed and so forth.

Example:

```
.BYTE   1.,20.      ;length=1,opcode for SET_PLANE_MASK
.WORD   ^B011       ;Enables GIDIS access to planes 1 and 2
                    ;Plane 3 is write-protected
```

6.52 SET_POSITION

SET_POSITION sets the new current position to the specified coordinates.

Opcode: 29 **Length:** 2

Format: SET_POSITION x, y

x	Specifies the new x coordinate of the current position.
y	Specifies the new y coordinate of the current position.

Status: SUCCESS

Notes:

- Current position may be set outside the clipping region. However, x and y should never be set larger than 16384 (2 to the 14th power). This will allow sufficient off-screen address space for accurate clipping.
- No drawing is done by the SET_POSITION instruction.

Example:

```
.BYTE   2.,29.  ;Length=2, opcode for SET_POSITION
.WORD   100.    ;New current position
.WORD   350.    ;is [100,350]
```

6.53 SET_PRIMARY_COLOR

SET_PRIMARY_COLOR sets the primary color index to use in drawing subsequent objects. Primary color is the color used for ON bits in current line texture, current area texture, and character glyphs.

Opcode: 21 **Length:** 1

Format: SET_PRIMARY_COLOR color-value

color-value	Specifies primary color as an index. On a multi-plane system, color-value functions as an index into the color map. On a single-plane system it specifies ON (color-value not 0) or OFF (color-value 0).

Status: SUCCESS

Notes:

- Refer to the INITIALIZE instruction for a list of the power-on default colors.
- If color-value is greater than color map size, color-value modulus map size is used.
- This instruction is affected by the SET_PLANE_MASK instruction.
- No drawing is done by the SET_PRIMARY_COLOR instruction.

Device Notes

- See Appendix E for the relationship between color and pens in Plotter GIDIS.

Example:

```
.BYTE   1.,21.  ;length=1,opcode for SET_PRIMARY_COLOR
.WORD   4.      ;defines primary color as color number 4
```

6.54 SET_REL_POSITION

SET_REL_POSITION sets a new current position as an offset from the old current position.

Opcode: 30 **Length:** 2

Format: SET_REL_POSITION dx, dy

dx	Specifies the x coordinate of the new current position as: x of current position + dx.
dy	Specifies the y coordinate of the new current position as: y of current position + dy.

Status: SUCCESS.

Notes:

- SET_REL_POSITION [dx,dy] is always the same as SET_POSITION [Current x + dx, Current y + dy].
- Current position may be set outside the clipping region. However, x and y should never be set larger than 16384 (2 to the 14th power). This will allow sufficient off-screen address space for accurate clipping.
- No drawing is done by the SET_POSITION instruction.

Example:

```
                        ;Current position is [100,350]
        .BYTE   2.,30.  ;Length=2, opcode for SET_REL_POSITION
        .WORD   100.    ;Relative position is
        .WORD   -50.    ;[+100,-50]
                        ;New current position is [200,300]
```

6.55 SET_SECONDARY_COLOR

SET_SECONDARY_COLOR sets the secondary color, for use in drawing subsequent objects. Secondary color is the color used for OFF bits in the current line texture, current area texture, and glyphs. It is also the color generated by NEW_PICTURE and ERASE_CLIPPING_REGION, and scrolled in by SCROLL_CLIPPING_REGION.

Opcode: 15 **Length:** 1

Format: SET_SECONDARY_COLOR color-value

color-value	Specifies secondary color as an index. On a multi-plane system, color-value functions as an index into the color map. On a single-plane system, it specifies ON (color-value not 0) or OFF (color-value 0).

Status: SUCCESS

Notes:

- Refer to the SET_COLOR_MAP_ENTRY description for a list of the power-on default colors.
- If color-value is greater than color map size, color-value modulus map size is used.
- SET_SECONDARY_COLOR is affected by the SET_PLANE_MASK instruction.
- This instruction does not draw anything or affect the view surface.

Device Notes

- See Appendix E for the relationship between colors and pens.
- Plotter GIDIS never changes the secondary color: the paper always remains the same color. However there is an effect. If you set secondary color to N, drawing in color 0 will draw with the pen that is normally used when drawing with color N.
- If secondary color modulus 16 is greater than or equal to 8, Plotter GIDIS slows pen speed to 10 cps. The slower speed results in better quality when drawing a transparency.

Example:

```
        .BYTE   1.,15.  ;length=1,opcode for SET_SECONDARY_COLOR
        .WORD   1.      ;defines secondary color as index 1
```

6.56 SET_WRITING_MODE

SET_WRITING_MODE defines how PRO/GIDIS interprets ON and OFF bits in line textures, area textures, and glyphs. There are 10 options as described in Table 6-11.

Opcode: 22 **Length:** 1

Format: SET_WRITING_MODE mode-code

mode-code — Specifies one of the integer values listed in Table 6-11.

Table 6-11: Writing Mode Options

Code	Writing Mode	Description
0	Transparent	Updates the current position, but does no drawing.
1	Transparent Negate	Updates the current position, but does no drawing.
2	Complement	If current-pattern-bit is on, complements the color of the current pixel. This means the current pixel is set to (2 ** plane's current color). In a 3 plane system, complementing 2 sets it to 6 (2 ** 3 - 2).
3	Complement Negate	If current-pattern-bit is off, complements the current pixel.
4	Overlay	If current-pattern-bit is on, the current pixel is set to the current primary color.
5	Overlay Negate	If current-pattern-bit is off, the current pixel is set to the current primary color.

Code	Writing Mode	Description
6	Replace	If current-pattern-bit is on, the current pixel is set to the current primary color (same as overlay). If current-pattern-bit is off, the current pixel is set to secondary color.
7	Replace Negate	If current-pattern-bit is off, the current is set to the current primary color. If current-pattern-bit is on, the current pixel is set to secondary color.
8	Erase	The current pixel is set to secondary color.
9	Erase Negate	The current pixel is set to primary color.

Status: SUCCESS if a valid mode is requested; otherwise, FAILURE.

Notes:

- No drawing is done by the SET_WRITING_MODE instruction.

Figure 6-9 shows the same line texture (which includes ON and OFF bits) drawn over light and dark areas in all visible writing modes.

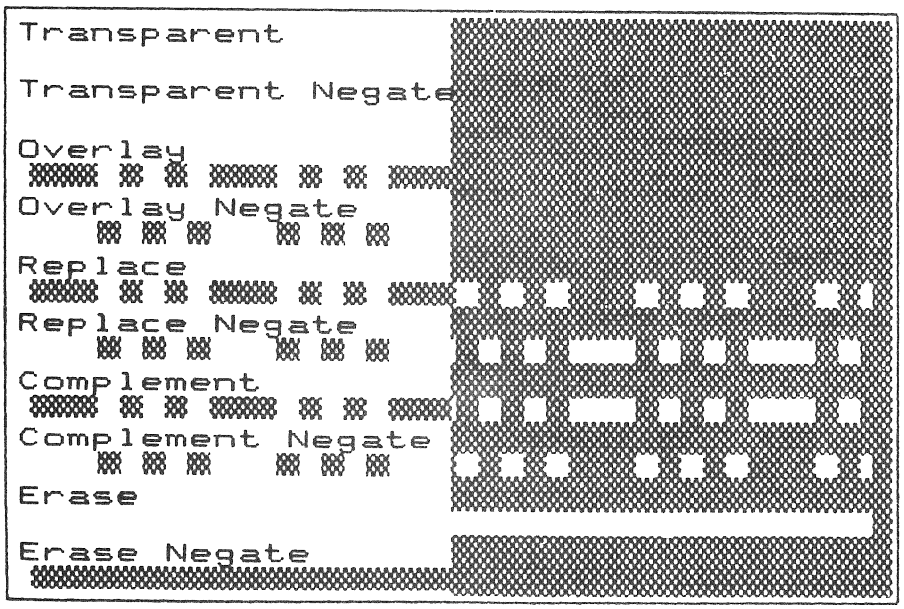


Figure 6-9: Writing Modes Shown with Line Texture

Device Notes

- Plotter GIDIS treats modes 2, 3, 4, 5, 6, 7 as overlay and modes 0, 1, 8, 9 as transparent.

Example:

```
        .BYTE   1.,22.  ;length=1,opcode for SET_WRITING_MODE
        .WORD   6.      ;Specifies REPLACE writing mode
```

APPENDIX A

PRO/GIDIS INSTRUCTION SUMMARIES

This appendix contains PRO/GIDIS instruction summaries and report tags for quick reference.

The instruction summaries are in two different formats: in ascending opcode order and in alphabetic order. The opcode and number of arguments are shown as separate byte values, and as opcode word values. The opcode word value = (opcode * 256) + number of arguments. Note, when the resultant opcode word value is greater than 32,000, it is subtracted from 65,536 (2**16) and a negative opcode word value results.

Table A-1: GIDIS Instructions in Opcode Order

Opcode	Number of Arguments	Opcode Word	Instruction and Arguments
0	0	0	NOP
1	1	257	INITIALIZE mask
3	2	770	SET_AREA_TEXTURE_SIZE w, h
4	4	1028	SET_OUTPUT_CLIPPING_REGION ulx, uly, w, h
5	6	1286	SET_OUTPUT_CURSOR alpha, index, w, h, ox, oy
6	0	1536	NEW_PICTURE

Opcode	Number of Arguments	Opcode Word	Instruction and Arguments
9	4	2308	SET_GIDIS_OUTPUT_SPACE x, y, w, h
12	2	3074	SET_OUTPUT_IDS w, h
13	4	3332	SET_OUTPUT_VIEWPORT ulx, uly, w, h
14	2	3586	SET_AREA_TEXTURE a, c
15	1	3841	SET_SECONDARY_COLOR color
16	6	4102	SET_COLOR_MAP_ENTRY m, color, r, g, b, mono
17	3	4355	SET_LINE_TEXTURE patlen, pattern, size
19	4	4868	SET_PIXEL_SIZE w, h, ox, oy
20	1	5121	SET_PLANE_MASK mask
21	1	5377	SET_PRIMARY_COLOR color
22	1	5633	SET_WRITING_MODE mode
23	3N	5888+3N	DRAW_ARCS x, y, angle
24	0	6144	END_PICTURE
25	2N	6400+2N	DRAW_LINES x, y
26	2N	6656+2N	DRAW_REL_LINES dx, dy
27	3N	6912+3N	DRAW_REL_ARCS dx, dy, angle
28	0	7168	FLUSH_BUFFER
29	2	7426	SET_POSITION x, y
30	2	7682	SET_REL_POSITION dx, dy
31	0	7936	BEGIN_FILLED_FIGURE
32	0	8192	END_FILLED_FIGURE
33	4 or 5	8448+N	BEGIN_DEFINE_CHARACTER c, w, nw, nh, [loff]

Opcode	Number of Arguments	Opcode Word	Instruction and Arguments
34	2+N	8706+N	LOAD_CHARACTER_CELL c, w, d0 . . .d15
35	N	8960+N	DRAW_CHARACTERS char-index
36	0	9216	END_DEFINE_CHARACTER
37	2	9474	LOAD_BY_NAME name_0, name_1
37	3-7	9472+N	LOAD_BY_NAME Ch1, Ch2, Ch3, ... Chn
38	1	9729	SET_ALPHABET alphabet
40	2	10242	SET_CELL_DISPLAY_SIZE w, h
41	2	10498	SET_CELL_EXPLICIT_MOVEMENT dx, dy
42	1	10753	SET_CELL_MOVEMENT_MODE flags
43	1	11009	SET_CELL_RENDITION flags
44	1	11265	SET_CELL_ROTATION angle
45	2	11522	SET_CELL_UNIT_SIZE w, h
46	5 or 6	11775+N	CREATE_ALPHABET w, h, extent, flags, [initialize], [ave-width]
48	0	12288	ERASE_CLIPPING_REGION
52	2	13314	SCROLL_CLIPPING_REGION dx, dy
53	3	13571	SET_OUTPUT_RUBBER_BAND type, x, y
54	0	13824	REQUEST_CELL_STANDARD
55	0	14080	REQUEST_CURRENT_POSITION
57	0	14592	REQUEST_OUTPUT_SIZE
58	0	14848	REQUEST_STATUS
65	1	16641	SET_CELL_OBLIQUE angle

Opcode	Number of Arguments	Opcode Word	Instruction and Arguments
69	2	17666	SET_AREA_CELL_SIZE w, h
71	0	18176	REQUEST_VERSION_NUMBER
72	1	18433	SET_OUTPUT_CURSOR_RENDITION mask
74	N	18944+N	DRAW_PACKED_CHARACTERS 2charindex
128	0	-32768	END_LIST
141	6 OR 7	-29434	PRINT_SCREEN x, y, w, h, hxly, dxly, [mask]
145	2	-28414	SET_OUTPUT_BITMAP bitmap-no, dis-flag

Table A-2 lists GIDIS instructions in alphabetical order.

Table A-2: GIDIS Instructions in Alphabetical Order

Opcode	Number of Arguments	Opcode Word	Instruction and Arguments
33	4 or 5	8448+N	BEGIN_DEFINE_CHARACTER c, w, nw,nh, [loff]
31	0	7936	BEGIN_FILLED_FIGURE
46	4,5 or 6	11775+N	CREATE_ALPHABET w, h, extent, flags [initialize], [ave-width]
23	3N	5888+3N	DRAW_ARCS x, y, angle
35	N	8960+N	DRAW_CHARACTERS char-index
25	N	6400+N	DRAW_LINES x, y
74	N	18944+N	DRAW_PACKED_CHARACTERS 2charindex
27	3N	6912+3N	DRAW_REL_ARCS dx, dy, angle
26	N	6656+N	DRAW_REL_LINES dx, dy
36	0	9216	END_DEFINE_CHARACTER
32	0	8192	END_FILLED_CHARACTER
128	0	-32768	END_LIST
24	0	6144	END_PICTURE
48	0	12288	ERASE_CLIPPING_REGION
28	0	7168	FLUSH_BUFFER
1	1	257	INITIALIZE mask
37	2	9474	LOAD_BY_NAME name_0, name_1

Opcode	Number of Arguments	Opcode Word	Instruction and Arguments
37	3-7	9472+N	LOAD_BY_NAME Ch1, Ch2, Ch3, ...Chn
34	2+N	8706+N	LOAD_CHARACTER_CELL c, w, d0, ...d15
6	0	1536	NEW_PICTURE
0	0	0	NOP
141	6 OR 7	-29434	PRINT_SCREEN x, y, w, h, hxly, dxly, [mask]
54	0	13824	REQUEST_CELL_STANDARD
55	0	14080	REQUEST_CURRENT_POSITION
57	0	14592	REQUEST_OUTPUT_SIZE
58	0	14848	REQUEST_STATUS
71	0	18176	REQUEST_VERSION_NUMBER
52	2	13314	SCROLL_CLIPPING_REGION dx, dy
38	1	9729	SET_ALPHABET alphabet
69	2	17666	SET_AREA_CELL_SIZE
14	2	3586	SET_AREA_TEXTURE a, c
3	2	770	SET_AREA_TEXTURE_SIZE w, h
40	2	10242	SET_CELL_DISPLAY_SIZE w, h
41	2	10498	SET_CELL_EXPLICIT_MOVEMENT dx, dy
42	1	10753	SET_CELL_MOVEMENT_MODE flags
65	1	16641	SET_CELL_OBLIQUE angle
43	1	11009	SET_CELL_RENDITION flags
44	1	11265	SET_CELL_ROTATION angle
45	2	11522	SET_CELL_UNIT_SIZE w, h

Opcode	Number of Arguments	Opcode Word	Instruction and Arguments
16	6	4102	SET_COLOR_MAP_ENTRY
9	4	2308	SET_GIDIS_OUTPUT_SPACE x, y, w, h
17	3	4355	SET_LINE_TEXTURE patlen, pattern, size
145	2	-28414	SET_OUTPUT_BITMAP bitmap-no,dis-flag
4	4	1028	SET_OUTPUT_CLIPPING_REGION ox, oy, w, h
5	6	1286	SET_OUTPUT_CURSOR a, c, w, h, ox, oy
72	1	18433	SET_OUTPUT_CURSOR_RENDITION
12	2	3074	SET_OUTPUT_IDS w, h
53	3	13571	SET_OUTPUT_RUBBER_BAND type, x, y
13	4	3332	SET_OUTPUT_VIEWPORT x, y, w, h
19	4	4868	SET_PIXEL_SIZE w, h, ox, oy
20	1	5121	SET_PLANE_MASK mask
29	2	7426	SET_POSITION x, y
21	1	5377	SET_PRIMARY_COLOR color
30	2	7682	SET_REL_POSITION dx, dy
15	1	3841	SET_SECONDARY_COLOR color
22	1	5633	SET_WRITING_MODE mode

Table A-3 lists report tags.

Table A-3: Report Tags

Tag Number	Argument Length	Opcode Word	Report Tag and Arguments
1	2	258	CURRENT_POSITION_REPORT x,y
2	9	521	OUTPUT_SIZE_REPORT ulx, uly, screen_width, screen_height, total_width, total_height, resolution_x, resolution_y, total_plane_mask
4	1	1025	STATUS_REPORT code
5	4	1284	CELL_STANDARD_REPORT uw, uh, dw, dh
7	2	1794	VERSION_NUMBER_REPORT code, version

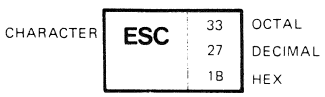
APPENDIX B

DEC MULTINATIONAL CHARACTER SET

ROW	COLUMN 0		COLUMN 1		COLUMN 2		COLUMN 3		COLUMN 4		COLUMN 5		COLUMN 6		COLUMN 7	
	b8 b7 b6 b5 BITS	0 0 0 0	0 0 0 1		0 0 1 0		0 0 1 1		0 1 0 0		0 1 0 1		0 1 1 0		0 1 1 1	
	b4 b3 b2 b1															
0	0 0 0 0	NUL	0 0 0	DLE	20 16 10	SP	40 32 20	0	60 48 30	@	100 64 40	P	120 80 50	`	140 96 60	p	160 112 70
1	0 0 0 1	SOH	1 1 1	DC1 (XON)	21 17 11	!	41 33 21	1	61 49 31	A	101 65 41	Q	121 81 51	a	141 97 61	q	161 113 71
2	0 0 1 0	STX	2 2 2	DC2	22 18 12	"	42 34 22	2	62 50 32	B	102 66 42	R	122 82 52	b	142 98 62	r	162 114 72
3	0 0 1 1	ETX	3 3 3	DC3 (XOFF)	23 19 13	#	43 35 23	3	63 51 33	C	103 67 43	S	123 83 53	c	143 99 63	s	163 115 73
4	0 1 0 0	EOT	4 4 4	DC4	24 20 14	$	44 36 24	4	64 52 34	D	104 68 44	T	124 84 54	d	144 100 64	t	164 116 74
5	0 1 0 1	ENQ	5 5 5	NAK	25 21 15	%	45 37 25	5	65 53 35	E	105 69 45	U	125 85 55	e	145 101 65	u	165 117 75
6	0 1 1 0	ACK	6 6 6	SYN	26 22 16	&	46 38 26	6	66 54 36	F	106 70 46	V	126 86 56	f	146 102 66	v	166 118 76
7	0 1 1 1	BEL	7 7 7	ETB	27 23 17	'	47 39 27	7	67 55 37	G	107 71 47	W	127 87 57	g	147 103 67	w	167 119 77
8	1 0 0 0	BS	10 8 8	CAN	30 24 18	(	50 40 28	8	70 56 38	H	110 72 48	X	130 88 58	h	150 104 68	x	170 120 78
9	1 0 0 1	HT	11 9 9	EM	31 25 19	)	51 41 29	9	71 57 39	I	111 73 49	Y	131 89 59	i	151 105 69	y	171 121 79
10	1 0 1 0	LF	12 10 A	SUB	32 26 1A	*	52 42 2A	:	72 58 3A	J	112 74 4A	Z	132 90 5A	j	152 106 6A	z	172 122 7A
11	1 0 1 1	VT	13 11 B	ESC	33 27 1B	+	53 43 2B	;	73 59 3B	K	113 75 4B	[	133 91 5B	k	153 107 6B	{	173 123 7B
12	1 1 0 0	FF	14 12 C	FS	34 28 1C	,	54 44 2C	<	74 60 3C	L	114 76 4C	\	134 92 5C	l	154 108 6C	|	174 124 7C
13	1 1 0 1	CR	15 13 D	GS	35 29 1D	-	55 45 2D	=	75 61 3D	M	115 77 4D	]	135 93 5D	m	155 109 6D	}	175 125 7D
14	1 1 1 0	SO	16 14 E	RS	36 30 1E	.	56 46 2E	>	76 62 3E	N	116 78 4E	^	136 94 5E	n	156 110 6E	~	176 126 7E
15	1 1 1 1	SI	17 15 F	US	37 31 1F	/	57 47 2F	?	77 63 3F	O	117 79 4F	_	137 95 5F	o	157 111 6F	DEL	177 127 7F

C0 CODES — GL CODES (ASCII GRAPHICS)

KEY



DEC MULTINATIONAL CHARACTER SET

8		9		10		11		12		13		14		15		COLUMN	
1 0 0 0		1 0 0 1		1 0 1 0		1 0 1 1		1 1 0 0		1 1 0 1		1 1 1 0		1 1 1 1		b8 b7 BITS b6 b5 b4 b3 b2 b1	ROW
	200 128 80	DCS	220 144 90		240 160 A0	°	260 176 B0	À	300 192 C0		320 208 D0	à	340 224 E0		360 240 F0	0 0 0 0	0
	201 129 81	PU1	221 145 91	¡	241 161 A1	±	261 177 B1	Á	301 193 C1	Ñ	321 209 D1	á	341 225 E1	ñ	361 241 F1	0 0 0 1	1
	202 130 82	PU2	222 146 92	¢	242 162 A2	2	262 178 B2	Â	302 194 C2	Ò	322 210 D2	â	342 226 E2	ò	362 242 F2	0 0 1 0	2
	203 131 83	STS	223 147 93	£	243 163 A3	3	263 179 B3	Ã	303 195 C3	Ó	323 211 D3	ã	343 227 E3	ó	363 243 F3	0 0 1 1	3
IND	204 132 84	CCH	224 148 94		244 164 A4		264 180 B4	Ä	304 196 C4	Ô	324 212 D4	ä	344 228 E4	ô	364 244 F4	0 1 0 0	4
NEL	205 133 85	MW	225 149 95	¥	245 165 A5	µ	265 181 B5	Å	305 197 C5	Õ	325 213 D5	å	345 229 E5	õ	365 245 F5	0 1 0 1	5
SSA	206 134 86	SPA	226 150 96		246 166 A6	¶	266 182 B6	Æ	306 198 C6	Ö	326 214 D6	æ	346 230 E6	ö	366 246 F6	0 1 1 0	6
ESA	207 135 87	EPA	227 151 97	§	247 167 A7	·	267 183 B7	Ç	307 199 C7	Œ	327 215 D7	ç	347 231 E7	œ	367 247 F7	0 1 1 1	7
HTS	210 136 88		230 152 98	¤	250 168 A8		270 184 B8	È	310 200 C8	Ø	330 216 D8	è	350 232 E8	ø	370 248 F8	1 0 0 0	8
HTJ	211 137 89		231 153 99	©	251 169 A9	1	271 185 B9	É	311 201 C9	Ù	331 217 D9	é	351 233 E9	ù	371 249 F9	1 0 0 1	9
VTS	212 138 8A		232 154 9A	ª	252 170 AA	º	272 186 BA	Ê	312 202 CA	Ú	332 218 DA	ê	352 234 EA	ú	372 250 FA	1 0 1 0	10
PLD	213 139 8B	CSI	233 155 9B	«	253 171 AB	»	273 187 BB	Ë	313 203 CB	Û	333 219 DB	ë	353 235 EB	û	373 251 FB	1 0 1 1	11
PLU	214 140 8C	ST	234 156 9C		254 172 AC	¼	274 188 BC	Ì	314 204 CC	Ü	334 220 DC	ì	354 236 EC	ü	374 252 FC	1 1 0 0	12
RI	215 141 8D	OSC	235 157 9D		255 173 AD	½	275 189 BD	Í	315 205 CD	Ý	335 221 DD	í	355 237 ED	ý	375 253 FD	1 1 0 1	13
SS2	216 142 8E	PM	236 158 9E		256 174 AE		276 190 BE	Î	316 206 CE		336 222 DE	î	356 238 EE		376 254 FE	1 1 1 0	14
SS3	217 143 8F	APC	237 159 9F		257 175 AF	¿	277 191 BF	Ï	317 207 CF	ß	337 223 DF	ï	357 239 EF		377 255 FF	1 1 1 1	15

C1 CODES — GR CODES (DEC SUPPLEMENTAL GRAPHICS)

KEY

CHARACTER — ESC — 306 OCTAL, 198 DECIMAL, C6 HEX

APPENDIX C

FONT FILE FORMAT

This Appendix describes the memory-resident format of a font file. A LOAD_BY_NAME font must have this format. GIDIS requires the data in a font file to be ordered as follows:

- header
- pointer table
- glyphs

C.1 HEADER

Header information (word wide) starts at the beginning of the font file. For example, Word 0 in the font is AL$MAG. Table C-1 shows the format of the header.

Table C-1: Header Format

Name	Offset	Description
AL$MAG	0	Magic number--must be 16473.
AL$STR	2	Structure version number--102.
AL$SIZ	4	Size of header in bytes--30.
AL$TOT	6	Total size of font file in bytes--may be up to 64KB.

Name	Offset	Description
AL$FLG	8	Flags--see CREATE_ALPHABET.
AL$RS0	10	Reserved for future.
AL$WID	12	Width of glyphs in this font--1 to 64 bits.
AL$HGT	14	Height of glyph--1 to 64 bits.
AL$FST	16	Index of first character represented in this font file--0 or greater.
AL$EXT	18	Extent of font file--number of glyph pointers you want in the font file. There is no specific limit if AL$TOT is less than 8KB; otherwise, AL$EXT must not be greater than 512.
AL$PTR	20	Offset from start of font file to pointer table. Pointer table must be present and on a word boundary. See Section C.2.
AL$RS1	22	Reserved for future use.
AL$FNT	24	Offset from start of font file to start of glyphs. See Section C.3.
AL$ORP	26	Offset from start of glyphs to out-of-range glyph, or -1. If -1, PRO/GIDIS will use its default out-of-range character.
AL$RS2	28	Reserved for future use.

C.2 POINTER TABLE

The pointer table contains AL$EXT words. Note that multiple table entries may point to the same glyph. If a table entry contains -1, GIDIS treats the character as if it were out of range. Table C-2 shows the format of the pointer table.

Table C-2: Pointer Table Format

Name	Description
1st entry	Offset from start of glyphs to the font information for the character with index AL$FST.
2nd entry	Offset from start of glyphs to the font information for the character with index (AL$FST + 1).
.	
.	
.	
Last entry	Offset from start of glyphs to the font information for the character with index (AL$FST + (n-1)).

C.3 GLYPHS

If AL$WID is 9 to 16, each glyph must start on a word boundary. Otherwise, glyphs may start on byte boundaries. There are no wasted bytes in a glyph. For example, if AL$WID is 22 (3 bytes per row of a glyph) and the glyph starts at offset x, then the second row starts at x + 3, the third row starts at x + 6, etc.

The leftmost pixel of a glyph is the low order bit of a row's first byte. Conversely, the rightmost pixel of a glyph is the first used bit of the row's last byte. For example, let AL$WID be 14, and examine the first row of a glyph. The leftmost pixel is bit 0 of byte 0 and the rightmost pixel is bit 5 of byte 1.

If the proportional flag is set, an extra word precedes other glyph data. The first byte of this extra word is the glyph's ave-width; the second byte is its left-offset.

APPENDIX D

MANAGING FONTS

D.1 MAKING A FONT AVAILABLE TO GIDIS

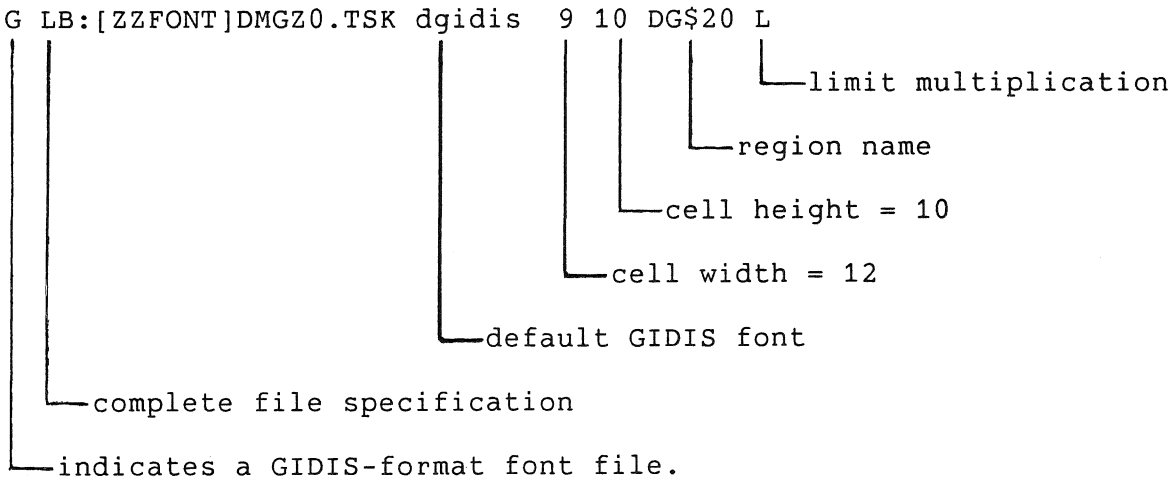
The .FDF files on LB:[ZZFONT] are files that tell the font server about the font files available on your system. When you boot your system, the font server is spawned and reads each .FDF file on [ZZFONT]. To make your fonts available in an application and for printing GIDIS files, have the application's installation file (.INS or .INB) copy the application's name with an .FDF extension (for example, APPname.FDF) to [ZZFONT].

An .FDF file contains one line per font file. Each line contains several fields. The fields are separated by spaces, but there may *not* be spaces before the first field in the line. You may use tabs in place of spaces. The order of fields is fixed. The fields must appear in the following order:

File type	A one-character field. It should be G for a GIDIS font file, and S for a DEC standard font file. S fonts are used only on the LN03.
File spec	The full file specification of the font file. There may be no embedded spaces.
Family ID	The font style. Note that a number of fonts are called DGIDIS, the family ID for the default GIDIS fonts.
Ave-width	For proportionally spaced fonts, the average width (number of horizontal pixels) of glyphs in a font file. For monospaced fonts, use the actual width (in pixels) of glyphs in this field. For example, the initial font on Video GIDIS has an ave-width of 12.

Character cell height	The height (number of vertical pixels) of glyphs in a font file. For example, the initial font on Video GIDIS has a height of 10.
Region name	The region name defined for the font when its .TSK file was built by calling GIFONT.
Rendition flags	Zero or more one-character fields. Each field defines the rendition built into the font. The defined options are I for italics, B for bold, P for proportional, and L for limit multiplication. Use L to prevent a heavily multiplied low detail font from being selected over a better font.

The following is a sample line in an .FDF file:



You may put blank lines in an .FDF file, but no comments.

D.2 FONT NAMING CONVENTIONS

A potentially large number of font files must coexist. For GIDIS fonts, this means their region names must coexist. Because region names are limited to six Radix-50 characters, not much name space exists. We suggest that you name fonts and regions as follows:

ffcwha

where

ff — Indicates the family ID of the font. For default GIDIS fonts, ff is DG.

c — Identifies the character set. The reserved values are $ for DEC Multinational (in file spec $ is replaced by M), P for patterns, and S for symbols.

wh — Specifies the ave-width and height of the font. The encoding for each is as follows:

X	means 7
Y	means 8
Z	means 9
0-9	means 10-19
A-K	means 20-30
L	means 32
M	means 34
N	means 36
.	
.	
.	
Z	means 60

NOTE

Because of the limitations of the Radix-50 naming space, some problems occur with this naming convention. You'll note that YZ can mean a character that is 8 x 9 or 58 x 60. In such cases, distinguish between the two by giving each size a unique family ID. Note also that you cannot create characters with odd numbered ave-widths or heights greater than 30, unless you create your own naming convention.

a indicates font rendition attributes. The following are reserved values:

B	for bold
C	for bold + italics
D	for bold + proportional
I	for italics
J	for italics + proportional
P	for proportional
Z	for bold + italics + proportional

For example the file name DGMFF.TSK and region name DG$FF indicate Default GIDIS, DEC Multinational, 20 x 20, and no rendition attributes.

D.3 FONTS SUPPLIED WITH GIDIS

Three different groups of fonts are supplied with GIDIS.

- Monospaced default GIDIS fonts that are automatically installed.
- Optional monospaced fonts that you can install.
- Optional proportionally spaced fonts that you can install.

Because font files require disk and system resources, only five font files from the default font family (DG) are loaded automatically. You may choose to load other monospaced and proportionally spaced font families as needed.

The following sections describe the font families and show an example of each. Each font family contains several font files. These font files contain several sizes of raster fonts and one stroke font.

D.3.1 Default GIDIS Fonts Loaded Automatically

Five default GIDIS (DG) fonts files are automatically installed. These fonts are monospaced, sans serif fonts that use the DEC Multinational Character Set.

Pro/Gidis V3.0 (Dgidis)

Figure D-1: Default GIDIS Monospaced Fonts

D.3.2 Rest of DGIDIS Monospaced Font Files

Installing the application "Rest of DGIDIS Monospaced Font Files" loads twelve additional font files from the default GIDIS font family. These files provide additional sizes of the same style font.

D.3.3 Proportionally Spaced Fonts

You can also install several proportionally spaced fonts.

When you install the application "Hershey Sans Serif Font," you load twelve sans serif font files that use the DEC Multinational Character Set.

Pro/Gidis V3.0 (Dgidis)

Figure D-2: Hershey Sans Serif Font

When you install the application "Hershey Serif Font," you load twelve serif font files that use the DEC Multinational Character Set.

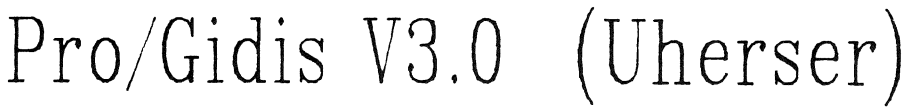


Figure D-3: Hershey Serif Font

When you install the application "Hershey Italicized Serif Font," you load twelve italicized serif font files that use the ASCII Character Set.

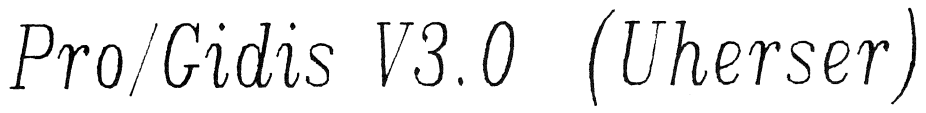


Figure D-4: Hershey Italicized Serif Font

When you install the application "Hershey Script Font," you load seven script font files that use the ASCII Character Set.

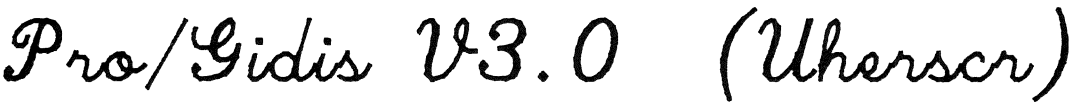


Figure D-5: Hershey Script Font

When you install the application "Hershey Gothic Font," you load five gothic font files that use the ASCII Character Set.

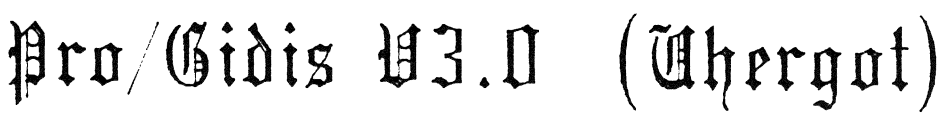


Figure D-6: Hershey Gothic Font

D.4 EDITING .FDF FILES

If you want to save resources, you can delete individual font files from .FDF files. For example, if you do not need certain sizes or do not need a stroke font, you can delete them from your .FDF files.

APPENDIX E

AREA TEXTURE AND COLOR ON THE PLOTTER

This appendix provides information on how the Hewlett-Packard HP7470A and HP7475A Plotters process GIDIS instructions differently from other supported devices. If an instruction is not mentioned, it performs as described in Chapter 4.

E.1 AREA TEXTURE

The plotter cannot handle bit patterned textures. Instead, area textures used for fill are mapped to a special set of hatch patterns. The mapping depends on the arguments supplied with SET_AREA_TEXTURE. There are three cases:

- Where alphabet = -1 and char-index = 0, the plotter draws horizontal hatch lines about .04 inches apart using the current linestyle.

- Where alphabet = 0 through 15 and char-index = 0, the plotter draws a true solid fill.

- Where alphabet = 0 through 15 and char-index is greater than 0, the plotter draws one of the hatch patterns shown in Table E-1.

Table E-1: Hatch Patterns for Char-Index 1 to 48

Solid Lines	Dashes	Long Dashes	Long/Short Dashes
Line Separation .06 inches			
1 plus sign	13 plus sign	25 plus sign	37 plus sign
2 slash	14 slash	26 slash	38 slash
3 horiz. line	15 horiz. line	27 horiz. line	39 horiz. line
4 backslash	16 backslash	28 backslash	40 backslash
5 vert. line	17 vert. line	29 vert. line	41 vert. line
6 X	18 X	30 X	42 X
Line Separation .11 inches			
7 plus sign	19 plus sign	31 plus sign	43 plus sign
8 slash	20 slash	32 slash	44 slash
9 horiz. line	21 horiz. line	33 horiz. line	45 horiz. line
10 backslash	22 backslash	34 backslash	46 backslash
11 vert. line	23 vert. line	35 vert. line	47 vert. line
12 X	24 X	36 X	48 X

The entire hatch pattern set repeats with codes 49 through 96. The basic 12 patterns are shown in Figure E-1.

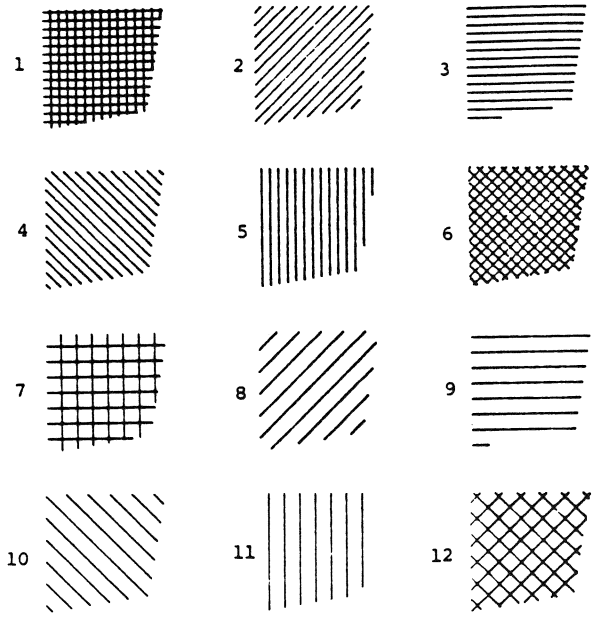


Figure E-1: Hatch Patterns 1 through 12

E.2 COLORS

You control the colors in a picture by placing pens in the carousel as desired. You can set up any pens you like. However, the recommended setting for the 6-pen plotter is:

Pen 1 - Red
Pen 2 - Green
Pen 3 - Blue
Pen 4 - Yellow
Pen 5 - Cyan
Pen 6 - Black

Because you control the colors, Plotter GIDIS ignores SET_COLOR_MAP_ENTRY. The 2-pen plotter handles colors as follows:

Color 0	background (no pen)
Color 1/5/7	left pen
Color 2/6	right pen
Color 3	left pen slowed down
Color 4	right pen slowed down

COLORS

The 6-pen plotter handles colors as follows:

Color 0	background (no pen)
Color 1	Pen 1
Color 2	Pen 2
Color 3	Pen 3
Color 4	Pen 4
Color 5	Pen 1 slowed down slightly
Color 6	Pen 5
Color 7	Pen 6

APPENDIX F

QUEUE I/O INTERFACE TO PRO/GIDIS FOR P/OS

Earlier versions of PRO/GIDIS used the P/OS Terminal Driver to access Video GIDIS through Queue I/O Request (QIO) and Queue I/O Request and Wait (QIOW) system directives. This appendix contains descriptions of the directive formats. QIO error messages are listed at the end of each description.

Figure F-1 depicts the instruction and parameter data path between your program and PRO/GIDIS.

You can use PRO/GIDIS from MACRO-11 or any supported Tool Kit high-level language that supports external MACRO-11 routines. The recommended method is to write callable MACRO-11 routines that issue QIO and QIOW directives. Tool Kit FORTRAN-77 provides its own callable QIO and WTQIO routines in SYSLIB.

For information on calling a MACRO-11 routine from one of the Tool Kit high-level languages, refer to the documentation for your programming language.

F.1 THE PRO/GIDIS INTERFACE

PRO/GIDIS instructions are sent to the graphics device with a QIO system directive that specifies the Write Special Data (IO.WSD) I/O function code. For low-overhead, high-speed, device interaction, a number of PRO/GIDIS instructions can be passed to the graphics device at one time. Status information returns with the Read Special Data (IO.RSD) I/O function call. P/OS transfers the instruction data to and from the graphics device according to the request priority and device availability.

Programs that use PRO/GIDIS also can use the Professional VT102 terminal emulator. Normal QIO directives (IO.WLB, IO.WVB, and so forth) are passed to the VT102 emulator. For more information, refer to the description of the Terminal Driver in the *P/OS System Reference Manual*.

THE PRO/GIDIS INTERFACE

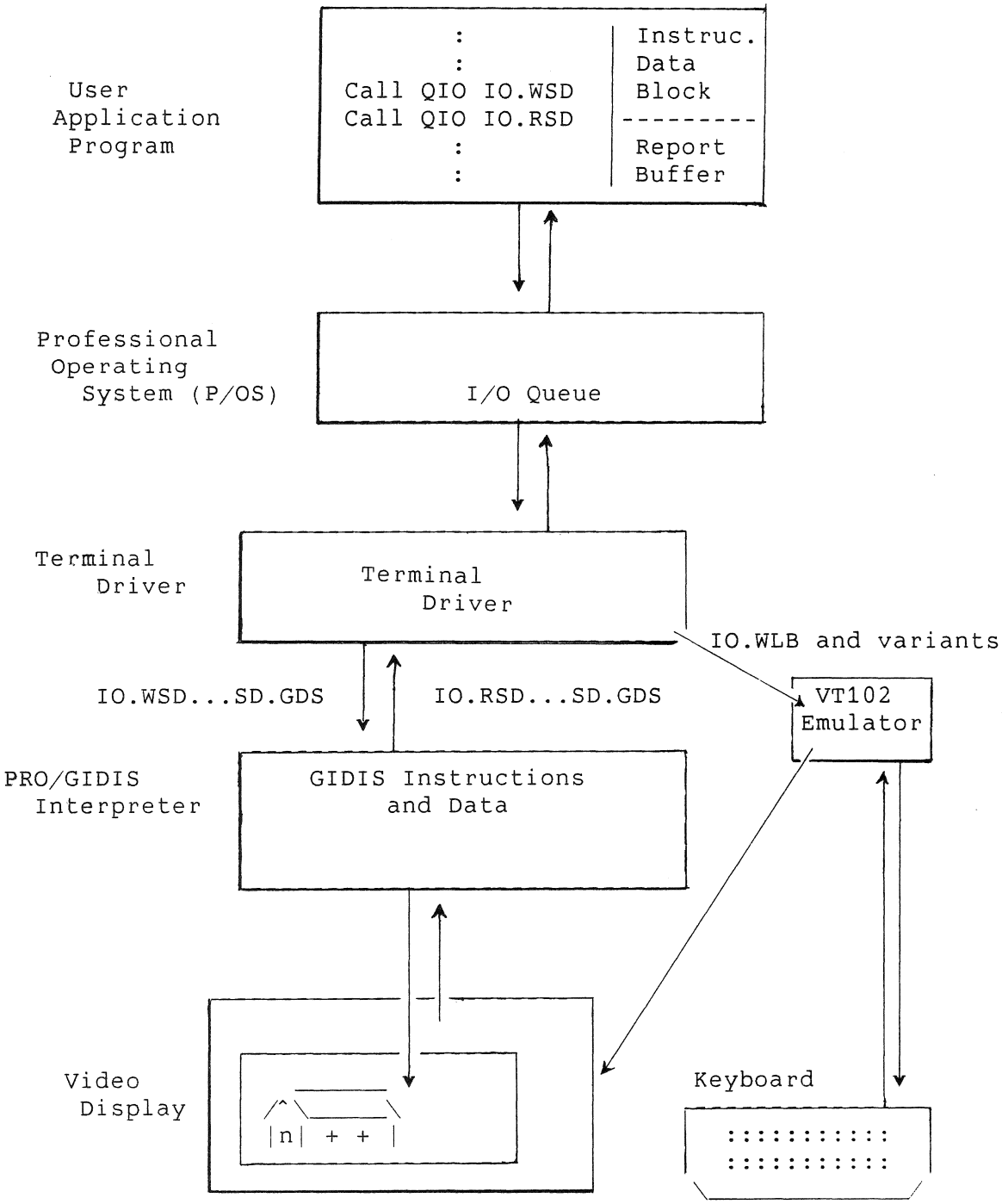


Figure F-1: PRO/GIDIS Data Path

On a single-plane system, both GIDIS and the VT102 emulator draw on the same plane and overwrite each other's data. On a three-plane system, the VT102 emulator only draws on plane three.

Thus, if PRO/GIDIS modifies only planes one and two, there will be minimal interference. The SET_PLANE_MASK instruction specifies which planes PRO/GIDIS can modify.

The VT102 emulator scrolls all three planes when the scrolling region is set to the entire screen. Any graphics information in any plane scrolls with the text. If the scrolling region is smaller than the entire screen, the VT102 emulator redraws the characters in their new positions. This does not scroll or otherwise affect graphics information in planes one and two but it erases graphics information in plane three.

You can send an RIS (Reset to Initial State - <ESC>c) escape sequence to the VT102 emulator in order to reset both the VT102 emulator and PRO/GIDIS to their initial states. PRO/GIDIS immediately performs an "INITIALIZE -1" instruction, clears the bitmap, and expects an opcode as the next word in the instruction/data stream. Thus, you can use RIS to ensure that your program and PRO/GIDIS are "in synch" when your program starts up. You cannot use it arbitrarily in the middle of picture generation because of the global initialization effect.

The QIO and QIOW directives are described in detail in the *P/OS System Reference Manual*. The examples in this appendix show the $S forms for clarity. The $C and $ forms can be used as well.

IO.WSD and IO.RSD are resolved in the normal manner for system symbols: the Application Builder gets them from module QIOSYM in SYSLIB.OLB. This works correctly in MACRO-11 but may not work with languages that have symbol naming restrictions. For example, FORTRAN-77 does not permit periods in symbol names.

F.1.1 Write Special Data (IO.WSD)

The write-special-data QIO function directs one or more instructions to PRO/GIDIS. The instructions and their associated parameter values are passed in a buffer that must have an even address.

The MACRO-11 format for the Write Special Data QIO (or QIOW) call is shown below.

NOTE

The punctuation marks and the items in bold are mandatory; nonbold items are optional. Items in uppercase letters must be used exactly as shown. Items in lowercase letters must be replaced as described.

QIOW$S #IO.WSD,LUN,efn,pri,isb,ast,**<buffer,length,,#SD.GDS>**

LUN Is a logical unit number assigned to the terminal.

efn Is an event flag number (required with the synchronous wait form QIOW).

pri Is the priority (ignored but must be present).

isb Is the address of the I/O status block.

ast Is the address of the AST service routine entry point.

buffer Is the address of the buffer containing PRO/GIDIS instructions and parameters.

length Is the length of the PRO/GIDIS instruction/parameter buffer (specified as an even number of bytes in the range 2 to 8128).

SD.GDS Is a data type parameter that indicates PRO/GIDIS output.

The QIO system directive returns status in a special global variable called $DSW. Some possible values are:

IS.SUC	Successful completion
IE.ILU	Invalid logical unit number
IE.IEF	Invalid event flag number

For a full list of error codes, refer to the QIO directive description and the terminal driver section of the *P/OS System Reference Manual*.

When the QIO directive is successful, it can return the following status codes in the I/O status block.

IO.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted
IE.DNR	Device not ready

F.1.2 Read Special Data (IO.RSD)

The read-special-data QIO function reads reports placed in the report queue by the following PRO/GIDIS report-request instructions:

- REQUEST_CURRENT_POSITION
- REQUEST_STATUS
- REQUEST_CELL_STANDARD

These instructions are detailed in Chapter 6.

The MACRO-11 format for the Read Special Data QIO or QIOW call is shown below.

NOTE

The punctuation marks and the items in bold are mandatory. Nonbold items are optional. Items in uppercase letters must be used exactly as shown. Items in lowercase letters must be replaced as described.

QIOW$S #IO.RSD,LUN,efn,pri,isb,ast,**<buffer,length,,#SD.GDS>**

LUN Is a logical unit number assigned to the terminal.

efn Is an event flag number (required with the synchronous wait form QIOW).

pri Is the priority (ignored but must be present).

isb Is the address of the I/O status block.

ast Is the address of the AST service routine entry point.

buffer Is the address of the buffer to contain PRO/GIDIS report data.

length Is the length of the PRO/GIDIS report buffer (specified as an even number of bytes in the range 2 to 8128).

SD.GDS Is a data type parameter that indicates PRO/GIDIS output.

If there is no data available, the QIO waits until enough data to fill the buffer becomes available. During this wait, no IO.WSD (write special data) is performed, even if the no-wait form was used. To avoid deadlock, the preferred method is to issue a QIO$W for the exact number of bytes expected after the request instruction is sent to PRO/GIDIS.

The QIO system directive returns status in a special global variable called $DSW. Some possible values are:

IS.SUC	Successful completion
IE.ILU	Invalid logical unit number
IE.IEF	Invalid event flag number

For a full list of error codes, refer to the QIO directive description and the terminal driver section of the *P/OS System Reference Manual*.

When the QIO directive is successful, it can return the following status codes in the I/O status block.

IO.SUC	Successful completion
IS.PND	I/O request pending
IE.ABO	Operation aborted
IE.DNR	Device not ready

F.2 PRO/GIDIS INSTRUCTION SYNTAX

The instructions syntax remains the same whether GIDCAL or QIO system directives are used to access PRO/GIDIS. See Chapter 3 for details.

F.3 SAMPLE MACRO-11 PROGRAM

```
IOSB:   .BLKW   2.
OBUF:   .BYTE   0.,55.  ;Length=0 REQUEST_CURRENT_POSITION
RBUF:   .BLKW   3.
                        ;SEND INSTRUCTION TO PRO/GIDIS
        QIOW$S #IO.WSD,#5,#1,,#IOSB,,<#OBUF,#2,,#SD.GDS>
        BCS     ERROR   ; DIRECTIVE FAILED
        TSTB    IOSB
        BLE     ERROR   ;OPERATION FAILED
                        ;READ THE REPORT
        QIOW$S #IO.RSD,#5,#1,,#IOSB,,<#RBUF,#6,,#SD.GDS>
        BCS     ERROR   ;BRANCH IF DIRECTIVE FAILED
        TSTB    IOSB
        BLE     ERROR   ;BRANCH IF OPERATION FAILED
                        ;
                        ; NEW CONTENTS OF RBUF:
                        ; BYTE AT RBUF    2.  (LENGTH)
                        ; BYTE AT RBUF+1  1.
                        ;   (CURRENT POSITION REPORT TAG)
                        ; RBUF+2:  CURRENT X POSITION
                        ; RBUF+4:  CURRENT Y POSITION
ERROR:                  ; Error handling routine
```

F.4 SAMPLE FORTRAN PROGRAM

```
      INTEGER*2   SDGDS, IOWSD, IORSD, ISSUC
      PARAMETER  (SDGDS=1), (IOWSD="5410), (IORSD="6030), (ISSUC=1)
      INTEGER*2  IOSB(2), IDS, OBUF
      INTEGER*2  RBUF(3),PARLST(6)

      OBUF = 55*256+0    !OPCODE 55=REQUEST_CURRENT_POSITION
                         !LENGTH=0


      CALL GETADR(PARLST(1),OBUF)          ! ADDRESS

      PARLST(2) = 2                        !LENGTH=2 BYTES
      PARLST(4) = SDGDS

      CALL WTQIO(IOWSD,5,1,0,IOSB,PARLST,IDS)
      IF (IDS.NE.ISSUC) GO TO 999          !DIRECTIVE FAILED
      IF (IOSB(1).NE.ISSUC) GO TO 999      !I/O REQUEST FAILED

      CALL GETADR(PARLST(1),RBUF)

      PARLST(2) = 6                 !EXPECTED LENGTH OF REPORT IN BYTES

      CALL WTQIO(IORSD,5,1,0,IOSB,PARLST,IDS)
      IF (IDS.NE.ISSUC) GO TO 999          !DIRECTIVE FAILED
      IF (IOSB(1).NE.ISSUC) GO TO 999      !I/O REQUEST FAILED

            ! NEW CONTENTS OF RBUF:
            ! RBUF(1):  258
            ! REPORT TAG = 1*256+2
            ! 1 = THE REPORT TAG AND 2 = LENGTH OF DATA FOLLOWING
            ! RBUF(2):  CURRENT X POSITION IN GIDIS OUTPUT SPACE
            ! RBUF(3):  CURRENT Y POSITION IN GIDIS OUTPUT SPACE

999         ! ERROR FOUND
```

APPENDIX G

GLOSSARY

The words in this glossary are used throughout this manual. These definitions are not absolute and might differ somewhat in other contexts. Where possible, the most common computer industry usage is the basis of the definition.

alphabet
A collection of characters. The character indexes are numbered 0,1,...n-1, where n is the extent of the alphabet.

anisotropic
An uneven ratio of width to height. In an anisotropic coordinate space, one unit in the X direction is not equal in size to one unit in the Y direction.

area texture
The two-dimensional binary pattern that you select to shade filled figures.

aspect ratio
The ratio of the width of an object to its height. Objects whose aspect ratio are important in graphics include video displays, pixels, and address spaces.

attribute
A property that tells GIDIS something about how to do the specified drawing operation. For example, when you tell GIDIS to draw a line, one of the attributes used in drawing the line is current primary color.

bitmap
The rectangular array of pixels (picture elements) that constitutes the view surface of a dot-oriented device. Also known as a raster or frame buffer. The Professional 350 has a bitmap 960 pixels wide by 240 pixels high.

character
A graphic symbol, such as a letter, number, or other typewritten symbol. In GIDIS, characters are elements in an alphabet. A character is uniquely identified by specifying its alphabet number and its index within the alphabet.

character cell
(See display cell or unit cell.)

clipping
Clipping means displaying only part of what is drawn. In GIDIS, you can select a clipping rectangle. What you draw inside the rectangle is displayed; what you draw outside the rectangle is not displayed.

color
In GIDIS the term has a double meaning. It has its usual "real world" meaning, and it also means an index into the GIDIS color map.

color map
A table whose entries contain a description of how to generate a particular color. In GIDIS this description is in terms of red, green and blue intensities. For example, bright yellow results from a maximum intensity of red, a maximum intensity of green, and a zero intensity of blue.

complement
The writing mode in which the foreground and background colors are reversed.

current position
The position in relation to which lines, arcs, and characters are drawn by GIDIS.

cursor
The marker displayed by GIDIS at the current position.

display cell
In GIDIS text processing, the display cell is that area of the view surface in which a unit cell is drawn. The top left corner of the unit cell is always placed at the top left corner of the display cell. Any portion of the display cell not covered by the unit cell is treated as though the unit cell is OFF for that area. If the unit cell is larger than the display cell, the unit cell is clipped at the display cell borders.

filled figure
To GIDIS, a figure is any sequence of connected lines and arcs. A filled figure is just a figure whose interior has been painted with the area texture of your choice.

font family
Loosely speaking, the style in which an alphabet is drawn, for example, Courier.

font file
The collection of glyphs and attribute information used by GIDIS to draw characters for a particular font.

GIDIS Output Space (GOS)
The isotropic coordinate space you set up for GIDIS to use in all drawing and report operations. A location within GOS maps to a location within your viewport.

global symmetry
Preservation of GIDIS Output Space relationships at the expense of Hardware Address Space relationships. For example, assume a ten-unit distance in GIDIS output space maps to 7.5 hardware pixels. With global symmetry, repeatedly moving ten GIDIS output space units results in a move of seven hardware pixels, then eight hardware pixels, then seven, and so forth. With local symmetry, repeatedly moving 10 GIDIS Output Space units always results in a move of 7 hardware pixels.

glyph
The data in a font file that tells GIDIS how to draw a particular character. In other words, it is the internal representation of a character.

Hardware Address Space (HAS)
The coordinate space (possibly anisotropic) used by a graphic output device. GIDIS hides this space from your program, and addresses the device's view surface through an isotropic Imposed Device Space.

image
A figure as defined in Imposed Device Space. In GIDIS you can display an image on a variety of output devices.

Imposed Device Space (IDS)
The isotropic coordinate space imposed on the device's view surface. You use IDS only to set the viewport. All other coordinates are in GIDIS Output Space (GOS).

isotropic
A 1:1 ratio of width to height. In an isotropic coordinate space one unit in the X direction is equal in size to one unit in the Y direction.

line texture
A linear pattern used to draw lines. Examples are solid, dashed, dotted, and so forth. PRO/GIDIS enables you to define any two-color pattern up to 16 units in length.

local symmetry
Preservation of Hardware Address Space relationships at the expense of GIDIS Output Space relationships. For example, assume a ten-unit distance in GIDIS output space maps to 7.5 hardware pixels. With local symmetry, repeatedly moving 10 GIDIS Output Space units always results in a move of 7 hardware pixels. With global symmetry, repeatedly moving ten GIDIS output space units results in a move of seven hardware pixels, then eight hardware pixels, then seven, and so forth.

origin
The origin of an address space is the point [0,0]. In PRO/GIDIS, the origin of IDS is always the upper left corner of the device's view surface. The origin of GIDIS output space is set by your program.

The origin of a character cell (either display cell or unit cell) is the point in the cell placed over the current position. This is also the point about which the cell rotates.

picture
A figure as defined in GIDIS Output Space. Once defined, you can store it in a file or map it to a viewport for display on a view surface.

pixel (picture element)
The smallest element of a view surface that can be assigned a color or intensity. In a single plane device, it is one bit in the bitmap.

pixel aspect ratio
The ratio of the width of a pixel to its height. The width is the horizontal distance between adjacent pixels, and the height is the vertical distance. Pixel aspect ratio is normally expressed as two small numbers, for example, 1:2. The pixel aspect ratio on the Professional 350 monitor is 2:5.

plane
A view surface that has N bits per pixel (that is, a pixel can be one of 2**N colors) is said to have N planes. A plane is a slice of a bitmap that contains one bit for each pixel.

primary color
The color index used to draw on-bits in area textures, line textures, and glyphs. GIDIS enables you to set the current primary color.

rubber band
A rubber band is a marker that shows the current position relative to a point of your choice, called the origin. There are two types of rubber bands available in PRO/GIDIS: the rubber band line and the rubber band rectangle. The line stretches from the rubber band origin to the current position. The rectangle has one corner at the rubber band origin and the opposite corner at the current position. The rectangle will degenerate to a line or point if the current position and rubber band origin are the same in one or both coordinates.

secondary color
The color index that indicates the absence of the drawing color. Thus its main function is to serve as the background color of a picture. In replace mode, it is also used to draw off-bits in area textures, line textures, and glyphs. GIDIS enables you to set the current secondary color.

standard display size
The standard display size is normally equal to the standard unit size. However, for alphabet 0 the standard display size is slightly smaller (horizontally) than the standard unit size. This is for increased compatibility with the VT125.

standard unit size
The size in GIDIS Output Space of a character such that 80 characters would fit horizontally and 24 characters would fit vertically, when IDS width/height is 8/5.

stroke device
A device whose view surface is written to with pen strokes, in contrast to a bitmap device, whose surface is written to with a sequence of dots.

text rendition
The variations of character appearance. For example, bolding and italics are renditions.

unit cell
In GIDIS, a character is viewed as a rectangular field of ON and OFF bits. ON bits form a character pattern; OFF bits form the background.

viewport
A rectangle within Imposed Device Space. You place this rectangle where you want the image to be displayed.

view surface
The part of the device upon which drawing can occur. For example, the screen is the view surface of the Professional Video monitor.

viewing transformation
The process of mapping graphic data from user coordinate space to display coordinate space.

window
A rectangle you define within GIDIS Output Space to control which part of your picture to map to a viewport.

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Please cut along this line.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or
Country

-- Do Not Tear - Fold Here and Tape ----------

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

Professional Workstations Publications
DIGITAL EQUIPMENT CORPORATION
146 Main Street, MLO21-2/T76
Maynard, Massachusetts 01754-2571

---- Do Not Tear - Fold Here ----------

Cut Along Dotted Line