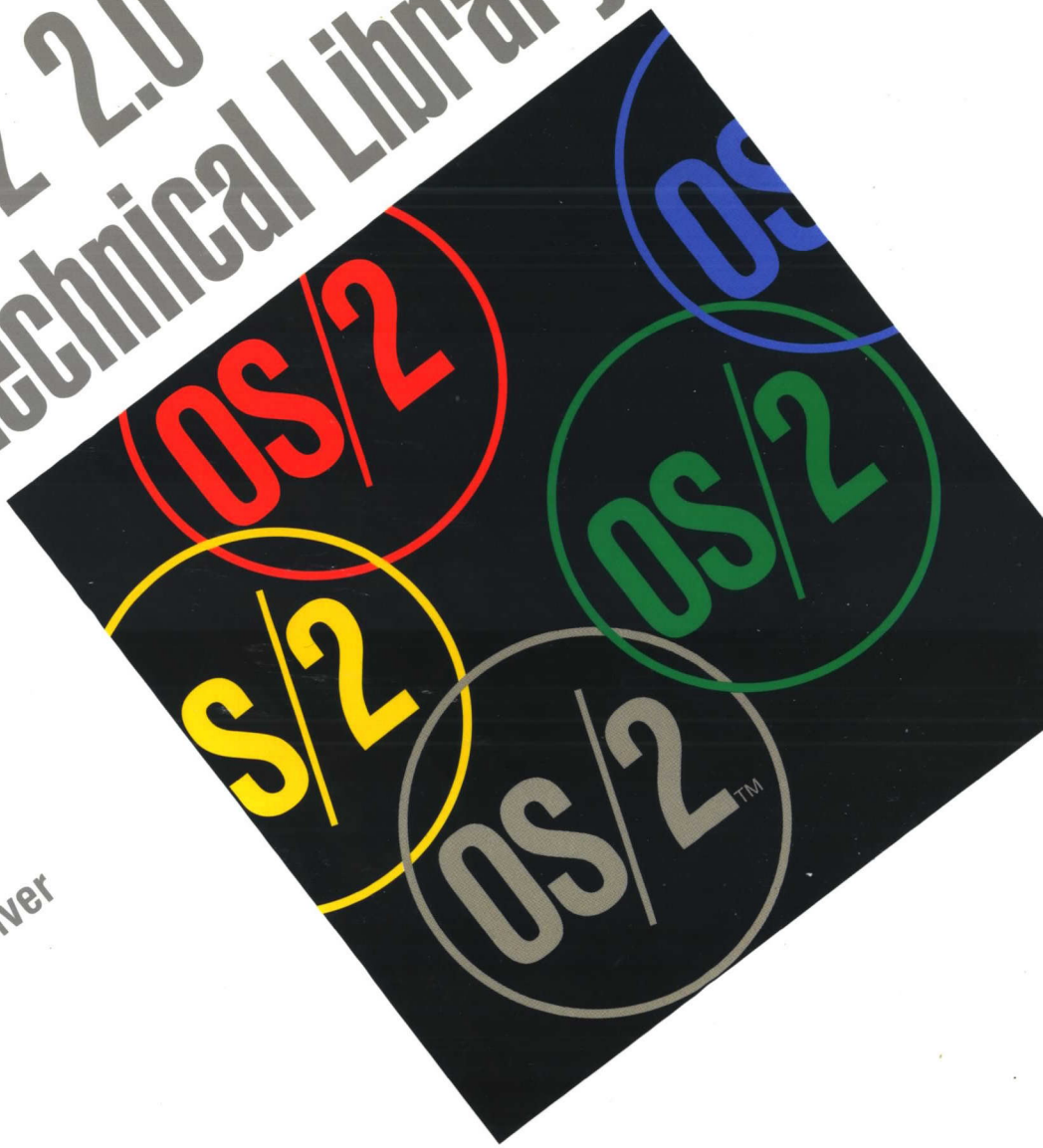
OS/2 2.0 Technical Library



Presentation Driver Reference

Version 2.00

IBM Programming Family

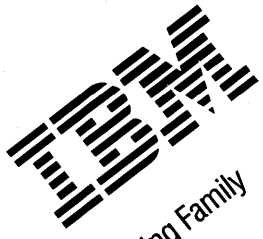
OS/2 2.0 Technical Library

THIS BOOK IS FOR PLANNING PURPOSES ONLY

Programmers who are interested in developing 32-bit presentation drivers for OS/2 2.0 and who are licensees of the IBM Developer's Toolkit for OS/2 2.0 should send in the form on page xiii to be included in the Presentation Driver Development Program. Members of this program will receive the pre-release items required for building 32-bit presentation drivers and will be notified of the distribution alternatives for presentation drivers developed under the program. This option is valid until the general availability of the pre-release code. IBM may change the terms of this option at any time without notice.

Presentation Driver Reference

Version 2.00



Programming Family

Note

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

First Edition (March 1992)

Contents

Part 1. Overview

Part 2. Development Considerations

Part 3. OS/2 2.0 Presentation Drivers

Part 4. Reference Material

Notices

Trademarks

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

IBM
OS/2
Presentation Manager
IBM Personal System/2

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

PostScript	Adobe Systems Incorporated
LaserJet	Hewlett-Packard Company
Microsoft	Microsoft Corporation
Windows	Microsoft Corporation

Double-Byte Character Set (DBCS)

Throughout this publication, there are references to specific values for character strings. These values are for the Single-Byte Character Set (SBCS). When using the Double-Byte Character Set, notice that one DBCS character equals two SBCS characters.

PRESENTATION DRIVER DEVELOPMENT PROGRAM FORM

To apply for membership in this program, you must be a licensee of the *Developer's Toolkit for OS/2 2.0*. Please enclose the original "Proof of License" card from your Toolkit with this completed form and return them to:

IBM Presentation Driver Development Program
Attention: M. Barovich, Internal Zip 1620
P.O. Box 1328
Boca Raton FL 33429-1328

This program is intended for developers who are planning to build 32-bit presentation drivers on OS/2 2.0. Members of this program will receive pre-release code that supports the development of presentation drivers as described in the publication *OS/2 2.0 Presentation Driver Reference*. This program will close when this code becomes generally available to users of OS/2 2.0.

Any information you provide on this form should be non-confidential to you or to any third party.

_____

Your Name

_____

Company Name (if applicable)

_____

Mailing Address

_____

_____

_____

Phone Number / FAX Number

Describe the presentation driver that you plan to develop:

_____

_____

_____

_____

_____

_____

About This Book

"OS/2," as used in this book, refers to Version 2.00 of the OS/2 operating system unless stated otherwise.

The *OS/2 2.0 Presentation Driver Reference* defines what a presentation driver is and how it operates. In addition, a description of the types of presentation drivers, their interfaces, and the available system services is provided. System programmers can use the information found in this book to write their own presentation drivers.

Detailed descriptions of control structures, data structures and I/O formats have been included where they are needed in order to understand and use the interfaces. For further information about structures and defined values, see the *OS/2 2.0 Presentation Manager Programming Reference* and the header files supplied with the *Developer's Toolkit for OS/2 2.0.*

Knowledge of at least one programming language that is used for writing OS/2 applications is necessary, and the programmer must be familiar with the workings of the OS/2 operating system, the Presentation Manager interface, and 80386 architecture. The programming concepts that should be understood before developing applications to run on OS/2 2.0 are found in the *OS/2 2.0 Application Design Guide.*

Note: Programmers who require information on 16-bit device drivers should refer to the *OS/2 2.0 Physical Device Driver Reference.*

This book consists of four parts: An introductory overview of presentation drivers (Part 1); an information section on development considerations (Part 2); a breakdown of the different types of OS/2 2.0 presentation drivers (Part 3); and a detailed reference section (Part 4). Also included are three appendixes covering syntax conventions, journal file format, and bitmap simulation. A more detailed description of the contents follows:

Part 1. Overview

Chapter 1. Presentation Drivers Overview
This chapter gives some general guidance on presentation drivers by indicating where they reside in the system and what they do.

Part 2. Development Considerations

Chapter 2. Design Considerations
This chapter describes general design considerations and specific design considerations for display and hardcopy presentation drivers.

Part 3. OS/2 2.0 Presentation Drivers

Chapter 3. Display Drivers
This chapter defines the entry points a display driver must export to the system so an application program can query the display driver and the system can enable it.

Chapter 4. Graphics Engine Hardcopy Drivers
This chapter describes hardcopy drivers, which use the graphics engine dispatch table, and the support functions that are provided by the spooler.

Chapter 5. Queue Drivers (Queue Processors)
This chapter describes the functions a queue driver must export so that they can be used by the spooler.

Chapter 6. Port Drivers
This chapter describes the functions a port driver must export so that they can be used by the Workplace Shell and spooler.

Part 4. Reference Material

Chapter 7. Exported Entry Points
This chapter describes the exported entry points, including the Enable entry point and all of its subfunctions.

Chapter 8. Mandatory Functions for All Drivers
This chapter describes the internal functions that must be supported by handling routines in the presentation driver. These functions must be supported by all presentation drivers.

Chapter 9. Mandatory Functions for Display Drivers
This chapter describes additional internal functions that must be supported by the display driver.

Chapter 10. Simulated Functions
This chapter describes internal functions that are supported in the operating system's graphics engine and can be hooked by the presentation driver in order to exploit special features in the device.

Chapter 11. Graphics Engine Internal Functions
This chapter describes internal functions that are supported in the graphics engine and can be called by the presentation driver.

Chapter 12. System Functions
This chapter describes system functions that can be called by a presentation driver but are not part of the OS/2 Application Program Interface (API).

Appendixes

Appendix A. Syntax Conventions
This appendix shows the conventions that have been used for the parameter names found in the Presentation Manager Library.

Appendix B. Journal File Format
This appendix shows the file format that the OS/2 graphics engine journal functions create in memory or on disk.

Appendix C. Bitmap Simulation (Hardcopy Drivers Only)
This appendix describes the bitmap simulation for hardcopy drivers.

A glossary and an index are included at the back of this book.

Part 1. Overview

Chapter 1. Presentation Drivers Overview

OS/2 operates within the confines of a four ring structure, Ring 0 through Ring 3. Ring 0 is the most protected ring and contains the OS/2 kernel, which is the main engine that *drives* the operating system. Ring 3 is the least protected ring and contains applications and system services. Ring 1 is not used by OS/2 2.0. Ring 2 is similar to Ring 0 in that it provides access to physical devices such as printers and displays. Ring 2 is similar to Ring 3 in that it is prohibited from modifying kernel data structures.

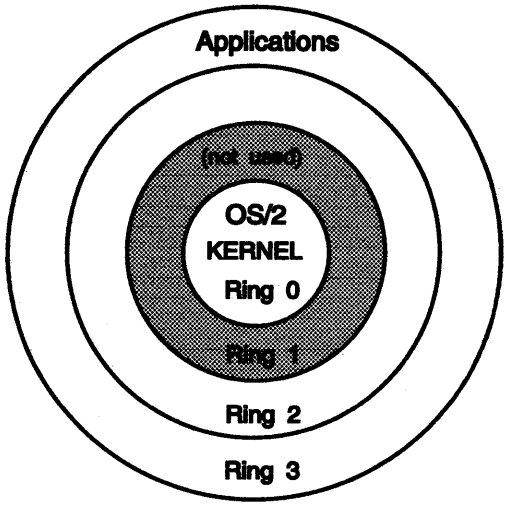


Figure 1-1. Ring Structure

Throughout the ring structure, functions can be called from the same ring level or any numerically lower ring level. Conversely, data can be accessed from the same ring level or any numerically higher ring level.

For the sake of compatibility between 16-bit and 32-bit code, all 32-bit code is treated as *Ring 2 Conforming*. This means that 32-bit code runs at the ring level of its caller, for example, running at Ring 3 if called from Ring 3 and running at Ring 2 if called from Ring 2. This allows easier access to functions and data, and eliminates many of the costly ring transitions encountered in the previous 16-bit system. To be Ring 2 Conforming (rather than strictly Ring 2 or Ring 3), a function cannot call any strictly Ring 3 code or directly access the hardware. That is, it must abide by the restrictions of both ring levels.

Many of the system services that exist at Ring 3 in dynamic link libraries (DLLs) access another DLL called PMGRE.DLL, which is the *graphics engine* or kernel of the OS/2 graphics subsystem. The graphics engine is composed entirely of Ring 2 Conforming code that interfaces with the presentation drivers, which interface directly with the physical device drivers (at Ring 0) and the hardware. These DLLs are identified as presentation drivers for hardcopy devices (*hardcopy drivers*) by the filename extension DRV, and as presentation drivers for display devices (*display drivers*) by the filename extension DLL. See Figure 1-3 on page 1-5.

The graphics engine loads and enables the presentation drivers, then dispatches calls to them through dispatch tables as Ring 2, Ring 3, or Ring 2 Conforming code. For optimal system performance, it is recommended that all of the functions in 32-bit presentation drivers be written as Ring 2 Conforming. This eliminates the need for ring transitions from system services at Ring 3 and from 16-bit presentation drivers at Ring 2.

By exporting a table named OS2_PM_DRV_RING_LEVELS in 32-bit presentation drivers, the ring level of each function call in the dispatch table can be selected.

Note: If this table is not exported, all 32-bit functions will be dispatched as Ring 2 Conforming.

The presentation driver has certain responsibilities to the graphics engine. Specifically, a number of entry points exist within the graphics engine that the presentation driver is required to *hook* (a mechanism by which procedures are called) and support. Many of these functions (as they currently exist in the graphics engine) are not truly functional, and if calls were made to these entry points, nothing would happen. In many cases, they simply return when called.

There are other entry points in the graphics engine that can *optionally* be hooked by the presentation driver (for example, where only light processing is required, it might be preferable to use the presentation driver). These entry points can be called by the presentation driver for special processing.

The graphics engine calls the entry points within the presentation driver by means of a *dispatch table*. The dispatch table is essentially a block of memory allocated by the graphics engine for the containment of entry points, and is assigned for use by a presentation driver. Each presentation driver loaded by the system is given its own separate dispatch table by the graphics engine. That is, when a presentation driver device context is enabled, the graphics engine allocates a dispatch table for that presentation driver and fills the dispatch table with pointers. Each entry in the table is a 32-bit pointer to a specific routine existing back in the graphics engine.

Because many of these routines must be hooked by the presentation driver, the graphics engine refers to the dispatch table to find the appropriate pointer any time that it calls a function in the presentation driver. At this point, however, all of the pointers in the dispatch table point back to routines in the graphics engine. The presentation driver must go into the dispatch table itself and replace some of the pointers with new pointers that point to corresponding routines within the presentation driver. This is mandatory for some routines, optional for others. The *hooking* of pointer entries in the dispatch table occurs the first time the presentation driver is called at its OS2_PM_DRV_ENABLE entry point for the first subfunction (see "Enable Subfunction 01H – FillLogicalDeviceBlock" on page 7-6). See Figure 1-2 on page 1-3.

The exported entry point OS2_PM_DRV_ENABLE (see page 7-3) has ten subfunctions. Four of these subfunctions are used in the enable process of a device context, three are used in the disable process, one is used to save the device context, one is used to restore the device context, and one is used to reset the device context.

When Enable Subfunction 01H – FillLogicalDeviceBlock is called, the graphics engine passes to it a pointer to the dispatch table that it allocated for the presentation driver. FillLogicalDeviceBlock then must substitute all of the mandatory pointers (and perhaps some optional routines) in the dispatch table with pointers to corresponding routines within the presentation driver itself.

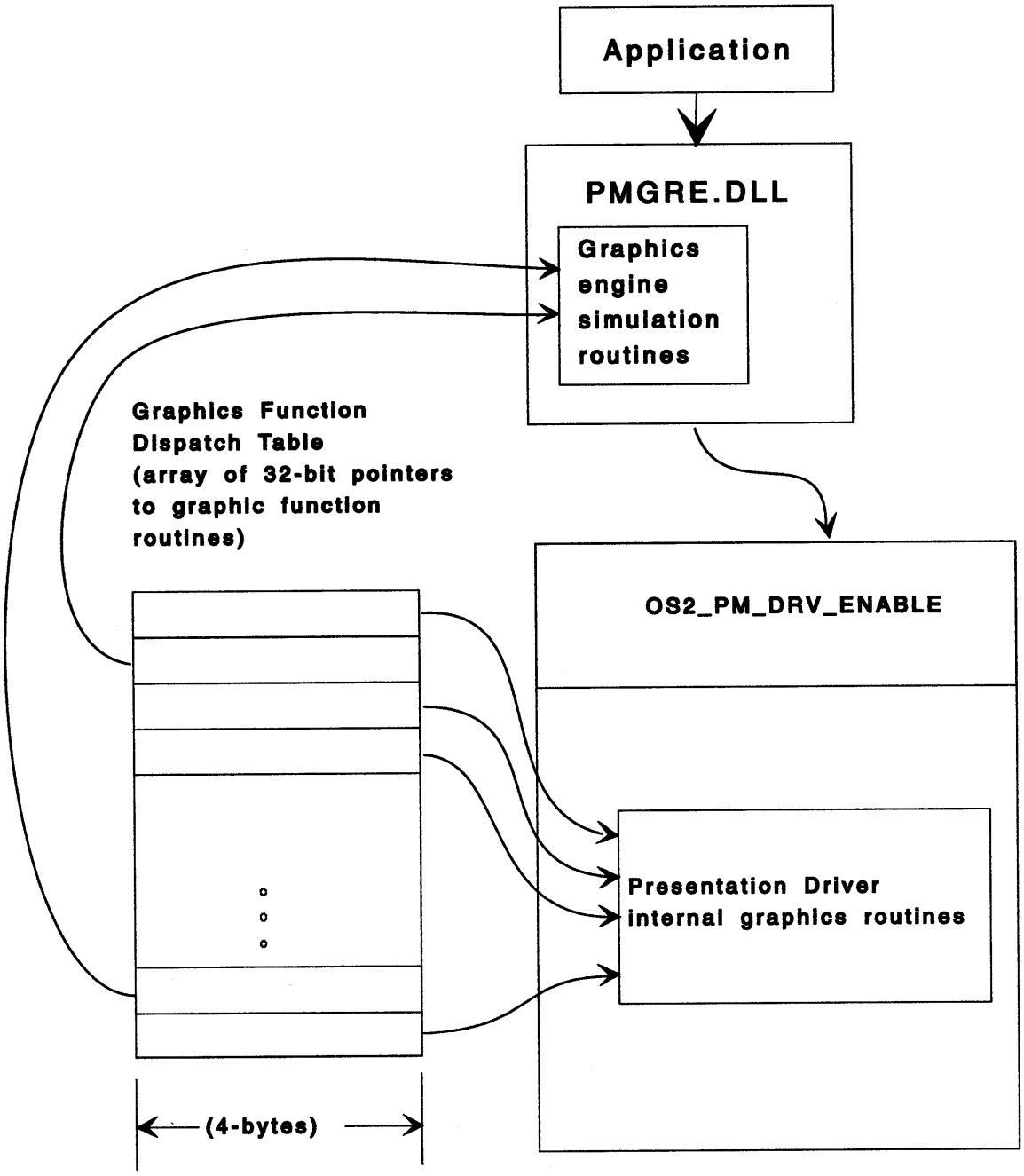


Figure 1-2. FillLogicalDeviceBlock Routine.

The *Developer's Toolkit for OS/2 2.0* provides support for writing source code in either C or Assembler. This support is comprised of a set of header files that define the functions, structures, and constants used on the internal interface to the presentation driver. Presentation drivers might also need to include OS2.H or OS2.INC, which define system functions, structures and constants.

As with other components of OS/2 2.0, the presentation driver architecture ensures that:

- Once loaded, the presentation driver can be enabled for use by multiple applications or processes.
- Once enabled, the presentation driver can support multiple instances of a device context for the owning application or process.

Notice that though many of the functions must be hooked by the presentation driver, some might also be passed back to the graphics engine for processing when called. It is recommended to save the original pointer values stored in the dispatch table by the graphics engine *before* hooking them. Having access to the original pointers to the graphics engine simulation routines allows the presentation driver to optionally make calls back into the graphics engine (that is, to have the graphics engine do the processing instead). This technique has often proven to be a tremendous time saver in developing OS/2 presentation drivers, especially when calling back to the graphics engine to perform clipping routines.

Display Devices

When the Presentation Manager* interface is initialized, the presentation driver for the attached display is loaded and enabled. This driver has direct access to the video hardware. The function calls passed to the presentation driver are processed and then passed to the adapter interface.

Hardcopy Devices (Printers and Plotters)

For hardcopy drivers, the presentation driver is loaded in response to an application or process calling DevOpenDC. Upon receipt of this routine, the Presentation Manager interface looks to see if the required presentation driver is loaded, and if it is able to handle the new device context (DC). If the presentation driver is loaded and can handle the DC, the driver is enabled for the new DC. If either of these conditions are not met, the required driver is loaded and then enabled.

* Trademark of the IBM Corporation

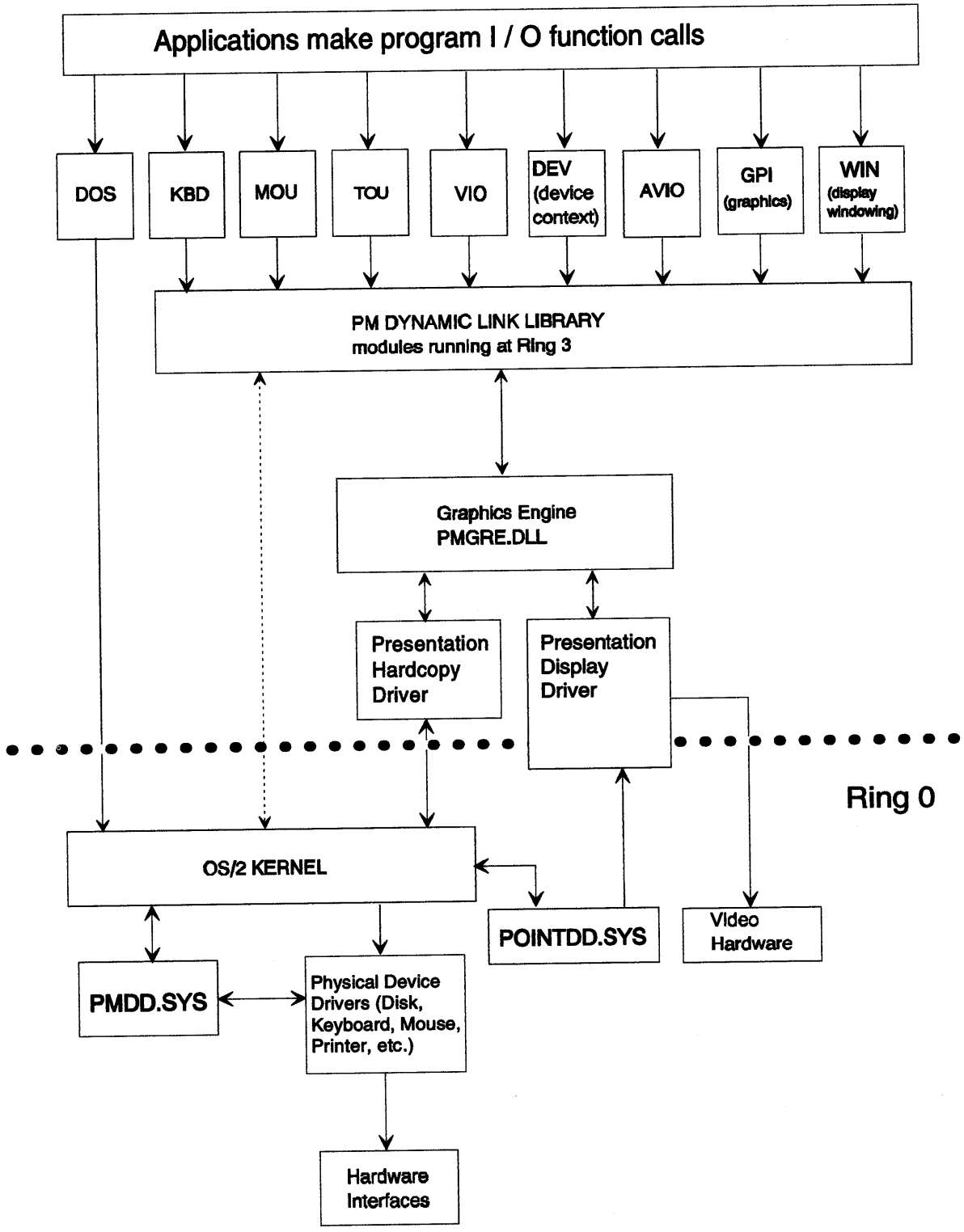


Figure 1-3. Presentation Drivers. Conceptual view of presentation drivers in the flow of control from an application program to the hardcopy device and the display screen.

Calling Conventions

Presentation drivers interact with both the external API, and the internal interface to the graphics engine and dispatch table. Parameters are passed as 32-bit values on the stack by using the C convention. These parameters are pushed to the stack in reverse order of how they appear in the statement. The hardware architecture locates the conceptual *top* of the stack at the high-order address of the stack space. When data is pushed to the stack, the stack pointer is decremented so that upon completion, the pointer addresses the first item of data.

At entry to the handling routine, the stack contains a frame of 32-bit parameters and a return address as shown in Figure 1-4.

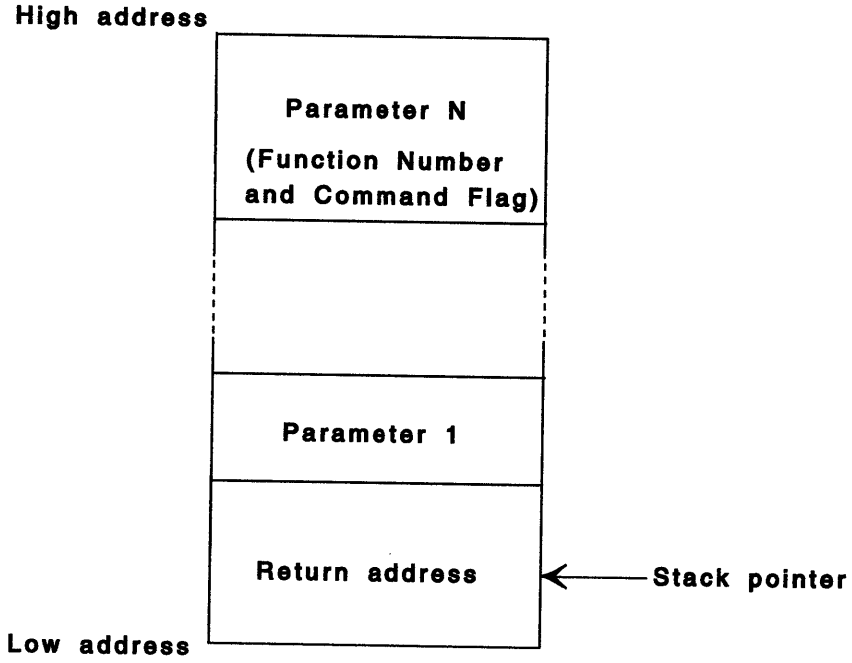


Figure 1-4. Stack Frame

Function Number and Command Flags

The first doubleword (DWORD) pulled from the stack contains two fields:

Field	WORD Type	Description
Function Number	Low-order	Identifies a specific function. Header files define symbolic names for the Gre*xxx* function numbers.
Command Flags	High-order	See below.

Command Flags The flags in this 16-bit field tell the presentation driver which operations need to be done while it processes the function:

COM_DRAW Bit 0. If this flag is set, the presentation driver must draw the output of the function at the device. When not set, the driver does not draw the output. The function must still be processed to update internal data such as current position and, if specified, boundary calculations.

COM_BOUND Bit 1. If set, the presentation driver must calculate the bounding rectangle for the output of the specified function. Upon completion, the driver calls its own GreAccumulateBounds routine to accumulate the bounding rectangle (GPI_BOUNDS in model space coordinates).

Note: All presentation drivers must be able to calculate bounds on any figure they can draw.

COM_CORR Bit 2. This flag is significant only for display drivers. If set, the display driver must determine whether the output of the specified function intersects the pick window. The result, TRUE or FALSE, is passed back to the caller in the return code for the called function. For details, see the function descriptions in Chapter 8, "Mandatory Functions for All Drivers" and Chapter 10, "Simulated Functions."

COM_ALT_BOUND Bit 3. This flag is significant only for display drivers. It indicates that the display driver should accumulate USER_BOUNDS in screen coordinates. Notice that user bounds are those used by the Window Manager. Hardcopy drivers do not accumulate user bounds.

COM_AREA Bit 4. If set, the function is part of an area. Calls to functions that define an area (for example, GreSetCurrentPosition and GrePolyLine), or that are invalid in an area definition, are passed back to the graphics engine for processing by the default handler.

If all functions in the area component have been hooked by the presentation driver, it is not necessary to pass back functions received with COM_AREA set.

COM_PATH Bit 5. This flag is similar to the COM_AREA flag. Calls to functions that define a path (for example, GrePolyLine and GreSetCurrentPosition), or that are invalid in a path definition, are passed back to the graphics engine for processing by the default handler.

If all functions in the path component have been hooked by the presentation driver, it is not necessary to pass back functions received with COM_PATH set.

COM_TRANSFORM Bit 6. If set, any coordinates given for the specified function must be transformed from world coordinates to device coordinates by using GreConvert. If not set, drawing functions expect or return screen coordinates and region functions expect or return device coordinates.

Note: For GreGetClipRects, if COM_TRANSFORM is set, device coordinates are returned. If it is not set, screen coordinates are returned.

COM_RECORDING Bit 7. This flag is ignored.

COM_DEVICE Bit 8. If set, the presentation driver must process the function. The driver should not pass the function to the graphics engine.

Note: COM_DRAW, COM_BOUND, COM_CORR, COM_ALT_BOUND, COM_AREA, and COM_PATH apply only to drawing functions. They are ignored by all other functions. The remaining bits of the Command Flags field are not defined and their values are ignored.

Device Context

Device contexts provide the mechanism that the application program uses to write output data to devices. The application, or one of its processes, opens a device context with DevOpenDC, associates a presentation space to the DC, and writes or draws in that space. Each DevOpenDC creates an instance of a DC. That instance is eliminated when the application closes the device context. The created DC is seen internally as a dispatch table. Calls from the application program to the DC are passed, as one or more internal Gre*xxx* routines, through the dispatch table to the handling routines in the presentation driver for the DC, or are passed back to the Gre*xxx* simulation routines.

Each instance of a device context has:

- Device context types
- Data types (for only OD_QUEUED device context type)
- Instance data
- Program stack

Device Context Types

The type of device to be opened is passed to the presentation driver when the device context is enabled:

OD_INFO The presentation driver does not generate any output. Information device contexts are used to retrieve information. All Gre*xxx* function routines passed to the presentation driver are processed as if the type were OD_DIRECT. In this way, the operating system can query statistics such as font metrics and boundaries.

OD_MEMORY The presentation driver processes the output data as for an OD_DIRECT device type except the output is written to a bit map that is compatible with the physical device. (The application program creates the bit map and associates it to the device context.)

OD_DIRECT The presentation driver processes the Gre*xxx* routines to generate device-specific output data. Hardcopy drivers use the file system Prt*xxx* interface to pass the output to the physical device driver. Display drivers use the adapter interface, for example, the IBM Personal System/2* Display Adapter 8514/A interface.

OD_QUEUED (Hardcopy drivers only.) The hardcopy driver opens a spool file and uses the spooler Spl*xxx* interface to send output to that file. For spooled output, the hardcopy driver must consider the DC data type. See Data Types (Hardcopy Drivers Only).

Note: OD_METAFILE and OD_METAFILE_NOQUERY are handled by PMGPI.DLL and are never passed to the presentation driver.

Data Types (Hardcopy Drivers Only)

For presentation drivers that support the device context type, OD_QUEUED, the driver must support the PM_Q_STD and PM_Q_RAW data types as defined by the Presentation Manager interface. Support for other data types is optional.

The concept of *data types* only applies when the device context type is OD_QUEUED. For all other device context types (OD_DIRECT, OD_MEMORY, OD_METAFILE, OD_METAFILE_NOQUERY, and OD_INFO), the *pszDataType* field in the DEVOPENSTRUC structure has no meaning. See "Enable Subfunction 02H – FillPhysicalDeviceBlock" on page 7-8 for details on when the DEVOPENSTRUC structure is passed to the presentation driver. However, a hardcopy driver is never requested to open an OD_METAFILE or

* Trademark of the IBM Corporation

OD_METAFILE_NOQUERY device context type because these types are handled by GPI. See Chapter 4, "Graphics Engine Hardcopy Drivers" for details on how the hardcopy driver creates the spool file.

The basic differences between data types, PM_Q_STD and PM_Q_RAW, are described below:

PM_Q_STD The hardcopy driver uses the spooler to create a device-independent spool file using the SplStd*xxx* and SplQm*xxx* interfaces.

PM_Q_RAW The hardcopy driver processes the Gre*xxx* functions to generate device-specific output data. This data is written using the spooler SplQm*xxx* interface to a spool file.

Instance Data

For every instance of a device context, the system has a doubleword that is reserved for use by the presentation driver. Typically, this doubleword is used by the presentation driver to hold a pointer to information about the current state of the device context. This pointer is returned to the system by the driver when the device context is enabled. On subsequent calls through the dispatch table, the pointer is passed back to the presentation driver as a parameter (pInstance) on the program stack. For more information, see "Enable Subfunction 05H — EnableDeviceContext" on page 7-12.

Program Stack

Presentation drivers can assume that a stack of 4KB is available for use when a function is passed to the driver at Ring 2. If it needs more than 4KB, the presentation driver should allocate its own stack space, switch to that stack on entry, and switch back to the original stack on exit. At Ring 3, the presentation driver will use the application's stack when a function is passed to the driver.

Function calls to the presentation driver use C calling conventions. Parameters are pushed to the stack in the opposite order as they are in the call statement.

Dynamic Link Library Functions

Functions exported and imported by a dynamic link library are identified in the library module definition file. These provide links between libraries and subsystems. For example, the components of the Presentation Manager interface must call an enable entry point in the presentation driver. The presentation driver needs access to the simulated functions in the graphics engine. For information on how to develop a dynamic link library (DLL), refer to the *OS/2 2.0 Programming Guide* and *OS/2 2.0 Application Design Guide*.

Note: The initialization routine for a dynamic link library, including presentation drivers, must be compiled to run at Ring 3 (privilege level 3).

Exported Functions

There are two types of exported functions used by OS/2 2.0:

- Presentation drivers
- Graphic engine functions.

Presentation Drivers: Dynamic link libraries for presentation drivers must export the appropriate set of the following entry points:

```
EXPORTS
   MoveCursorBlock     @103              /* Display drivers only  */
   OS2_PM_DRV_QUERYSCREENRESOLUTIONS     /* Optional              */
   OS2_PM_DRV_DEVMODE                    /* Hardcopy drivers only */
   OS2_PM_DRV_DEVICENAMES                /* Hardcopy drivers only */
   DrvInstall                            /* Optional              */
   DrvRemove                             /* Optional              */
   OS2_PM_DRV_RING_LEVELS                /* All drivers           */
   OS2_PM_DRV_ENABLE_LEVELS              /* All drivers           */
   OS2_PM_DRV_ENABLE                     /* All drivers           */
```

In addition to the entry points listed above, hardcopy drivers should export entry points for the routines that handle dialogs with the user. See "Exported Entry Points" on pages 3-1, 4-1, and 7-1.

Graphics Engine Functions: The graphics engine exports its own set of entry points. Those that are significant to the presentation driver are:

```
EXPORTS
   GETDRIVERINFO    @30
   SETDRIVERINFO    @31
```

GETDRIVERINFO: Used by the presentation driver to get the instance pointer, pInstance, for a specified device context, or to get a pointer to the bit-map header for a specified bit map. See Chapter 12, "System Functions" for more information.

SETDRIVERINFO: Used by the presentation driver to set a specified value in the instance pointer of a specified device context.

Note: Instance pointers (pInstance) and data are discussed under "Enable Subfunction 05H — EnableDeviceContext" on page 7-12.

Imported Functions

To call a Gre*xxx* function that is supported as a simulation or internal function in the graphics engine, call the imported entry point with the Gre*xxx* function parameters. The pInstance parameter should be NULL. For example, to call GreCreateJournalFile with the name assigned in the module definition file, use:

```
result = GreCreateJournal (pszFileName, flOption, cSize, 0L, NGreCreateJournalFile);
```

Note: NGreCreateJournalFile is defined in the header file. See the description of GreCreateJournalFile on page 11-12.

Simulations (presentation driver interface functions that are supported by handling routines in the graphics engine) can also be called at the addresses given in the default dispatch table. Use the addresses contained in the dispatch table that is passed to the presentation driver at Enable time.

Presentation Driver Interface

The internal Presentation Driver Interface (PDI) is comprised of a set of graphics engine (Gre*xxx*) functions that are called through a dispatch table. A *dispatch table* is an array of pointers to function handling routines. The low-order byte of the function number identifies the member of the array that contains the pointer for the function. The functions called through the dispatch table fall into three main groups:

- Functions that all presentation drivers must support. See Chapter 8, "Mandatory Functions for All Drivers."
- Functions that must be supported by display drivers. See Chapter 9, "Mandatory Functions for Display Drivers."

- Functions that are supported by simulations in the graphics engine. See Chapter 10, "Simulated Functions."

The first instance of a loaded presentation driver is given a copy of the default dispatch table. The Enable routine in the presentation driver modifies this copy so that, for those functions supported in the driver, the pointers address the function-handling routines of the presentation driver.

When the function is called a second time (or any time thereafter) for the same presentation driver, a NULL dispatch table pointer can be given because the graphics engine already has the table correctly initialized. Therefore, it is not necessary to reinitialize the table.

Part 2. Development Considerations

Chapter 2. Design Considerations

The following list contains design considerations for all presentation drivers:

- Angles
- Bounds computations
- Clipping
- Closing figures in areas and paths
- Coordinate values
- Positions within text functions
- Return codes
- Transform matrix values.

Angles

Angles are passed as signed 32-bit numbers. Zero refers to the direction of the positive x-axis; 360 represents 360°. Positive values represent counterclockwise angles from the positive x-axis.

Bounds Computations

All presentation drivers must accumulate bounds for unclipped primitives. Application bounds (COM_BOUND) are accumulated in model space. User bounds (COM_ALT_BOUND) are accumulated in device-coordinate space.

Clipping

The presentation driver must perform clipping for drawing and text functions, except for GreDrawLinesInPath and GrePolyShortLine. Clipping for these two functions is done by the graphics engine. The minimum requirement is to render each primitive clipped to a single rectangle and to clip each rectangle in turn. The rectangles can be enumerated by using "GreGetClipRects" on page 10-57.

Note: Rectangles might not always be valid. See "Drawing to Display Devices."

Closing Figures in Areas and Paths

The graphics engine generates closure lines for figures within areas and paths unless the presentation driver has opted to hook all the path and area functions. In this case, the presentation driver is responsible for closing any figures. For details, see "Area and Path Functions" on page 10-1.

Coordinate Values

All coordinates are passed to the presentation driver as 32-bit values. Unless stated otherwise, these values represent world coordinates. The graphics engine function, GreConvert, can be called to convert coordinates from one type to another. Coordinates must be converted back to world coordinates before returning to the presentation driver. Notice that *screen coordinates* are device coordinates to which the DC origin has been added.

Positions Within Text Functions

When positions are used, a text function takes the position from the base line of the text box. *Descenders* such as the tail of a lowercase *y* are expressed as a negative value relative to the base line.

Return Codes

The presentation driver must always return a full 32-bit (LONG) value. For example, BOOLEAN TRUE and FALSE are defined as:

```
#define TRUE   (1L);
#define FALSE  (0L);
```

Transform Matrix Values

Transform matrix elements are represented in fixed point notation, that is, as a 16-bit signed integer and a 16-bit fractional part. These precision limits apply during graphics engine matrix multiplication for all initial, intermediate, and final matrix element values.

Allocating Memory

Presentation drivers can allocate and manage memory by using:

1. A Dos*xxx* function such as DosAllocMem.
2. The SS*xxx* functions described in Chapter 12, "System Functions" on page 12-1.

Display drivers, or presentation drivers that wish to share objects such as bit maps and regions, always use the SS*xxx* functions to allocate memory for these objects. Memory allocated through calls to these functions is *shared memory* controlled by the memory allocator component of the graphics engine. Ownership of the memory can be transferred, or (when the owning DC ceases to exist) marked as having no owner.

Error Strategy

Presentation drivers support the error strategy implemented by the Presentation Manager interface. When an error occurs, the driver calls WinSetErrorInfo (see page 12-7) to log the appropriate error code and set the return code to show that an error was detected.

The component that implements a function must provide error checking for the environment, objects, and resources associated with it. The presentation driver needs to cater for:

- Fail-safe on routines that set attributes and transformation values. Any routine that changes attributes or transformation values must be able to restore the initial values if an error occurs during the change.
- Full error checking on symbol sets, fonts, bit maps, and regions.
- Segment drawing, drawing primitives, and primitive attributes in draw mode, unchecked parameters are passed directly to the graphics engine or the presentation driver. When one of these functions is hooked by the presentation driver, the handling routine must do the necessary error checking and log any errors, or reset any invalid values to their defaults, as appropriate.
- Any function with coordinates as parameters, the presentation driver must check that the values passed are valid. When an invalid coordinate is detected, the handling routine must log an error or use a default coordinate value, as appropriate.

For any defined error, the application sees the same error code regardless of whether the error was logged by the Graphics Programming Interface (GPI), graphics engine, or presentation driver.

Severity

Four severity levels are defined for error messages:

- Warning
- Error
- Severe error
- Irrecoverable error.

Warning: Function detected a problem, took remedial action, and was able to complete successfully.

Error: Function detected a problem for which no sensible remedial action is possible. The function is not executed and the system remains at the same state as when the function was requested.

Severe Error: Function detected a problem from which the system cannot reestablish its state. The function has partially executed and the application must now make some corrective action to restore the system to some known state.

Irrecoverable Error: Function detected an error from which it is impossible for the system to reestablish the state that it held at the time that the function was called. It is also impossible for the application to restore the system to some known state.

Presentation Manager Error Codes

Error codes are defined in the header file. These codes fall into two groups, general and specific. General error codes that are appropriate to many Gre*xxx* functions include:

Error Code	Must be logged by:
PMERR_COORDINATE_OVERFLOW	Functions requiring matrix computation
PMERR_INSUFFICIENT_MEMORY	Functions resulting in memory allocation
PMERR_INV_HBITMAP	Functions with *hbm* as an explicit or implicit parameter
PMERR_INV_HRGN	Functions with *hrgn* as an explicit or implicit parameter
PMERR_INV_COORDINATE	Functions with coordinate parameters
PMERR_INV_IN_AREA	Functions that are invalid inside an open area bracket
PMERR_BASE_ERROR	Functions that directly or indirectly issue DOS routines
PMERR_DEV_FUNC_NOT_INSTALLED	Functions not supported by the presentation driver

Specific error codes listed in the descriptions of each Gre*xxx* function are found in Chapter 8, "Mandatory Functions for All Drivers" through Chapter 11, "Graphics Engine Internal Functions."

To set an error code and the error's severity, the presentation driver must call WinSetErrorInfo. See Chapter 12, "System Functions" All error codes are listed and explained in the *OS/2 2.0 Presentation Manager Programming Reference*.

Exit List Processing

An *exit list* is a list of routines that are given control when the current process ends, normally or abnormally. The following is an example of exit list processing:

1. When the presentation driver's Enable subfunction 01H – FillLogicalDeviceBlock is called, the driver can call function, DosExitList:

   ```
   #define ROUTINE_ORDER 0x1000
   usResult = DosExitList (EXLST_ADD|ROUTINE_ORDER, (PFNEXITLIST)MyExitProc);
   ```

   This adds the function, MyExitProc, to the list of functions that are called when this process terminates (either normally or because of some error such as a GP fault).

2. When MyExitProc is called, the presentation driver can perform any necessary cleanup such as releasing any semaphores. The last call in MyExitProc is another call to DosExitList:

   ```
   usResult = DosExitList (EXLST_EXIT, (PFNEXITLIST)MyExitProc);
   ```

   This allows the operating system to transfer control to the next function in the list of Exit List processing functions for the process that has terminated. For more information, refer to DosExitList in the *OS/2 2.0 Control Program Programming Reference* and *OS/2 2.0 Programming Guide*.

At Enable time, the presentation driver must place an entry in the exit list for the application or process that opens the DC. This entry is a pointer identifying the routine in the presentation driver that releases all resources owned by the DC.

Note: When writing a presentation driver, consider what would happen if another thread of the process were to terminate.

Interrupts

Presentation drivers never use the CLI and STI macro assembler instructions because these instructions can interfere with some of the base OS/2 system operations.

Protecting Objects or Device Contexts

A process which attempts to use a locked object will return an error such as PMERR_HDC_BUSY. Although a device context is owned by a single application or process, the owner can access the device context (DC) through multiple threads. The presentation driver must provide a mechanism whereby a DC can register that it is busy and block access from other threads. In its simplest form, this is performed by the EnterDriver and LeaveDriver routines, which are called at the start and end of each function-handling routine in the presentation driver.

An example of a typical EnterDriver routine for a display driver is as follows:

```
/********************************** Typical EnterDriver Routine ***********************************/

enter_driver()
{
  do {                                  /* Lock DC for exclusive use of the current thread          */
       SemEnter(Device);                /* Some functions do not pass a Device Context (DC) handle  */

       if (hdc == NULL)
           return(SUCCESS);             /* Check validity of the passed DC handle                   */

       if (hdc == ERROR) {
           WinSetErrorInfo (SEVERITY_ERROR, PMERR_INV_HDC);
           SemLeave(Device);
           return(ERROR);
                          }             /* DC region must be validated before driver draws into it. */

       if (hdc_is_not_dirty)            /* Test the HDC_IS_DIRTY flag.  If the flag is set, the DC  */
           return(SUCCESS);             /* must be recalculated by the system.                      */

       SemLeave(Device);                /* Unlock DC.  Call back to engine to force DC calculation. */
       VisRegionNotify(hdc);            /* Loop back to reset lock and recheck.                     */
      } while (TRUE);
  }
```

Design Considerations for Display Drivers

When an application requires a user to choose an object from a presentation space, the user typically selects the object by positioning the mouse cursor over the object and clicking the mouse buttons. This action sends a message to the application, informing the program of the current (x, y) position of the mouse cursor. However, it is still up to the application to determine the object selected. This is accomplished by the application defining a rectangular area named the *pick aperture*, which is centered on the reported mouse position, and determining which, if any, of the currently defined segments intersect or lie completely within the pick aperture. The process of determining intersection or inclusion within the pick aperture is called *correlation*.

Correlation

Correlation must be performed by all display drivers in page coordinates on fully-clipped primitives. (Correlation is not required for hardcopy devices.) Correlating on areas is particularly complex because GreSetCurrentPosition and GreEndArea generate a closure line when the current position is not at the start of the current closed figure. This closure line can cause a correlation hit. Also, the area interior itself can cause a correlation hit that must be reported on the GreEndArea order.

The lines (arcs, full arcs, boxes, and fillets) defining the area boundary can cause a correlation hit if the area is specified with boundary. This hit must be reported when the function is issued. This means that other work must be done in addition to journaling the functions that define the area boundary.

Correlation and Retained Segments: To be a candidate for correlation, a retained segment must:

- Have a unique identifier
- Be a non-dynamic object
- Be defined as detectable (see following explanation).

Each primitive or group of primitives within a given segment must be capable of maintaining tag information. *Tag information* is added to an object in response to an application calling GpiSetTag(). The *pick tag* is a positive integer. If *0* is specified as the pick tag, detectibility should be turned *off* for

subsequent primitives. The pick tag is *0* by default. The tag specified in the most recent call to GpiSetTag() is the current tag, and remains in effect for all subsequent drawing operations until another call to GpiSetTag() is made. The tag is considered to be a part of the current attributes for the segment, and therefore is affected by the current attribute mode of the segment.

An application can request correlation data for:

- Segments that have been defined as both detectable and visible
- All non-zero segments that intersect the pick aperture regardless of the object detectability and visibility attributes.

The presentation driver returns the names of segments within the pick aperture in reverse order of their occurrence on the segment chain. This data is returned in the form of segment and tag pairs. Each unique segment and tag pair within the pick aperture is termed a *correlation hit.* Should two or more primitives within the current pick aperture have the same tag, they are considered as a single correlation hit. When a called segment is picked, correlation data is also returned for all segments above it in the hierarchy (up to and including the root segment). This also constitutes a single correlation hit.

Correlation and Nonretained Segments: *Nonretained graphics* are those objects that are correlated on *during* the drawing process. If nonretained objects are to be correlated on, they must have unique identifiers and be defined as detectable. The application must set the correlate bit of the Draw Control flag passed to the engine from the function GpiSetDrawControl().

Drawing to Display Devices

Because changes on the screen can affect more than one DC, the graphics engine notifies the display driver when a change occurs. Notification is performed through a call to the GreInvalidateVisRegion function (see page 9-9) in the display driver. This routine identifies the affected DCs and supplies a pointer (pInstance) to the instance data of each DC. The handling routine for GreInvalidateVisRegion sets a flag such as HDC_IS_DIRTY in the instance data for all identified DCs.

All routines that draw on the screen test the HDC_IS_DIRTY flag. If this flag is set, the routines call "VisRegionNotify" on page 12-6 before drawing the visible region.

Design Considerations for Hardcopy Drivers

For information on hardcopy drivers written to the graphics engine dispatch table, see Chapter 4, "Graphics Engine Hardcopy Drivers," Chapter 7, "Exported Entry Points," Chapter 10, "Simulated Functions," Chapter 11, "Graphics Engine Internal Functions," and Chapter 12, "System Functions"

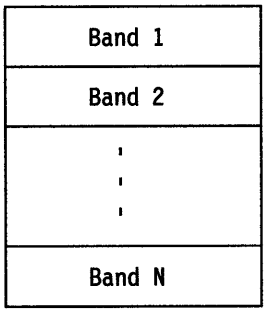
The following list contains design considerations that apply to only hardcopy drivers:

- Banding
- Document processing
- Extended attributes
- Hardcopy device names
- Hardcopy driver migration
- Hardcopy driver output to file
- Help
- Job error dialog

Banding

Banding is another technique that is available to presentation drivers for raster technology hardcopy devices. *Banding* means to break up a whole page into two or more bands (or strips) of raster data, which is recorded in memory as a bit map, and is then sent to the physical device or spooler. It is used to reach a balance between memory requirements and performance. This technique uses the graphics engine's journaling functions to save and replay a journal file of the Gre*xxx* calls for a whole page.

The hardcopy driver handles the output page as a number of bands and creates a bit map large enough for one band at a time:



The DC origin of the bit map is manipulated so that it relates to each band in turn. The hardcopy driver replays the journal file as many times as necessary to write into each band. Notice that Band 1 cannot be written into while the Gre*xxx* calls are being journaled. This is because the Command flag, COM_DRAW, is turned *off* between calls to GreStartJournalFile and GreStopJournalFile unless the JNL_DRAW_OPTIMIZATION flag is passed in on the call to function GreCreateJournalFile(). The hardcopy driver is told not to perform any output while the Gre*xxx* calls are being journaled unless the JNL_DRAW_OPTIMIZATION bit flag is set.

The size (width and height) of each band is determined by each hardcopy driver, dependent upon the type of physical device to which the output is to be sent and the amount of memory the hardcopy driver can use to build its bit map. As an example, a color laser printer might need the full 24-bits per pel, in which case, several bands might be needed to make a page. A simple dot matrix printer that uses 1-bit per pel could treat the whole page as a single band.

The data for each band is sent as a single band to the physical device for an OD_DIRECT device context, or to the spooler for an OD_QUEUED device context with a data type of PM_Q_RAW. Notice that the number of bytes of data that is sent might not be the same as the number of bytes required to create the bit map for a given band. This can be due to compression algorithms, which might be implemented in a given hardcopy driver, and understood by the firmware of a given physical device.

The technique of banding is performed by recording all of the graphics for a whole page in a journal file. When the journal file is complete (that is, the hardcopy driver has received either a DEVESC_NEXTBAND, DEVESC_NEWFRAME, or DEVESC_ENDDOC escape), the hardcopy driver plays the journal file, reprocesses the calls recorded to produce each band in turn, and clips the graphics recorded in the journal file to each band output. After all bands are output, the journal file is deleted. This banding technique uses the graphics engine journaling functions to save and replay a journal file of the Gre*xxx* calls. These graphics engine journaling functions are documented in "Journaling Functions (Hardcopy Drivers Only)" on page 11-1.

Each page of output is handled as a separate entity. The GreEscape routine for DEVESC_STARTDOC opens a journal file for the first page and registers it in the DC instance data. When GreEscape DEVESC_NEWFRAME or DEVESC_ENDDOC is received, the hardcopy driver writes the bands and closes the journal. If the escape code was DEVESC_NEWFRAME, the GreEscape routine opens and registers the journal file for the next page.

Figure 2-1 on page 2-8 gives an overview of how the presentation driver performs:

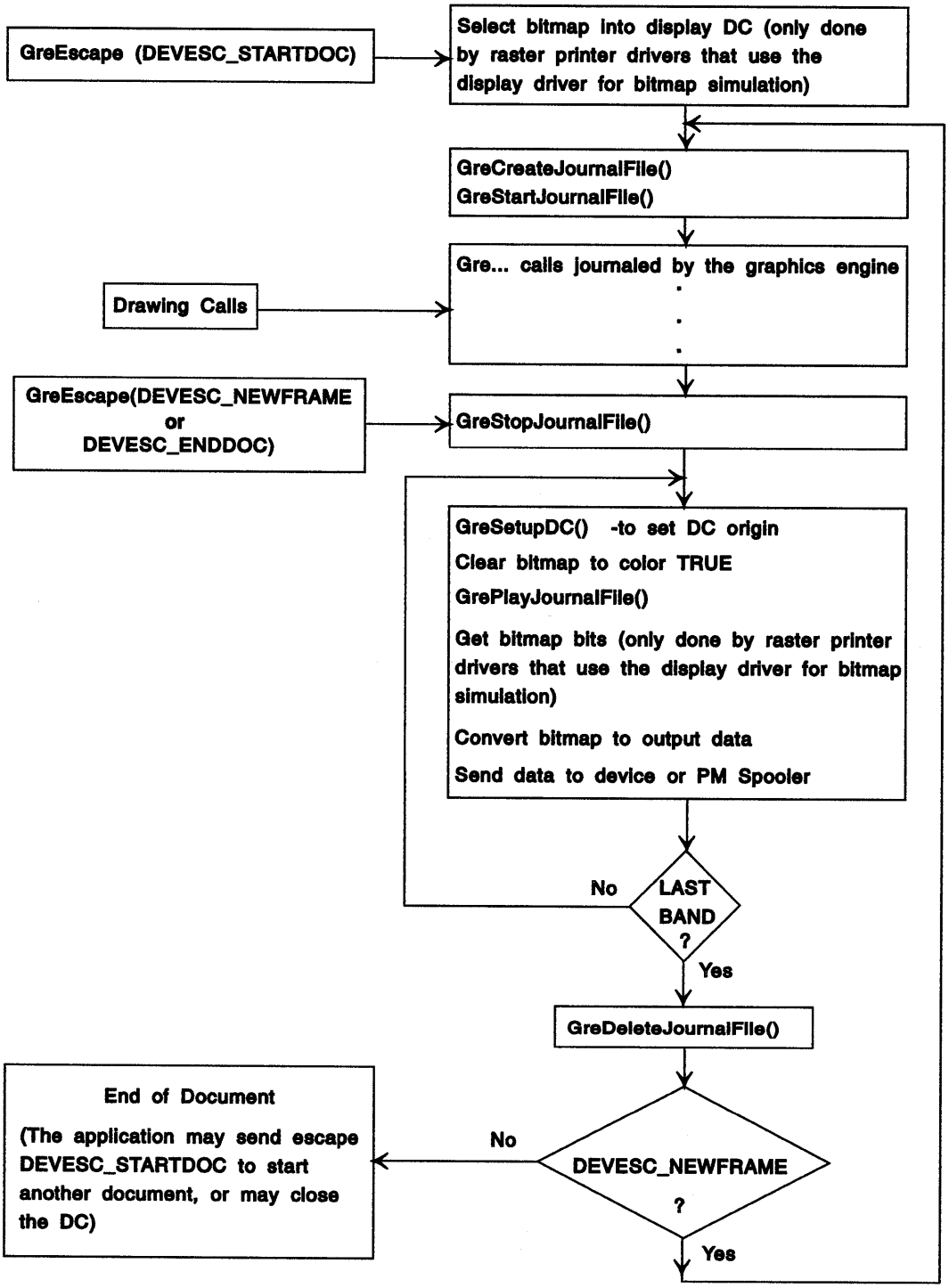


Figure 2-1. Overview of Presentation Driver Performance

Document Processing

Figure 2-2 on page 2-9 diagrams the state transitions for the GreEscape function calls used during document processing. The GreEscape function names are abbreviated for ease of reading. For example, GreEscape (DEVESC_STARTDOC) is shown as STARTDOC. See "Spool File Creation" on page 4-15 for details on each of the output actions.

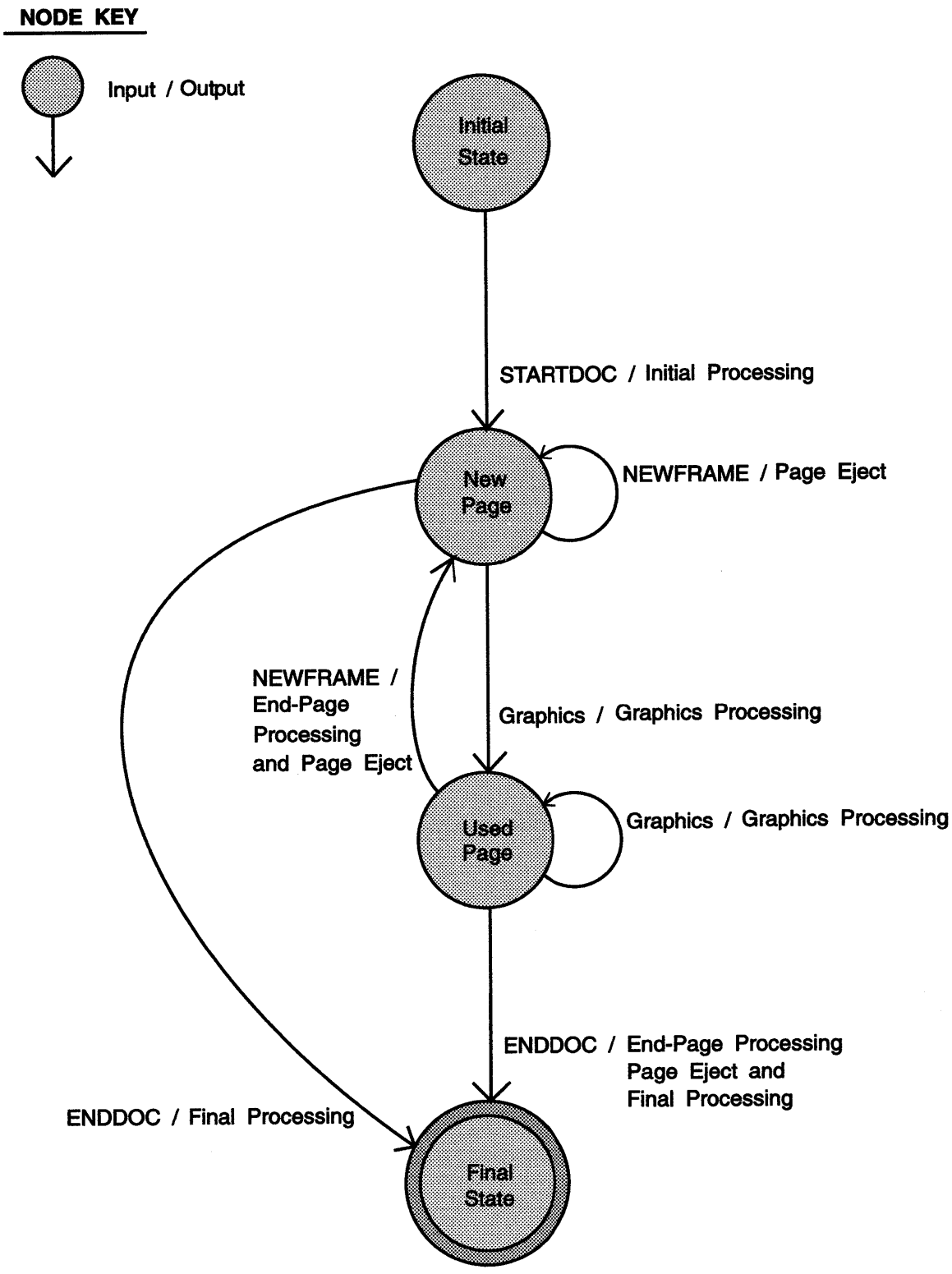


Figure 2-2. State Transitions for Document Processing by Hardcopy Drivers

Extended Attributes

Extended attributes (EA) are used for correct installation and operation of multi-file drivers. They are also used for improved performance during installation.

Extended attributes serve three purposes:

- Installation of multi-file drivers
- Version numbering
- Improved performance of drivers in the Workplace Shell.

Names are *comma* separated when more than one name is defined in an extended attribute. The examples that follow are for the HP LaserJet** hardcopy driver.

Installation of Multi-File Drivers: The following extended attributes can be used for the installation of multi-file drivers:

- **VENDORNAME (Optional)**

  Vendor name of the supported printers. For example, `VENDORNAME=HP`.

- **REQUIREDVENDORFILES (Optional, unless the driver requires other files)**

  Names of extra files required for correct operation by the driver files in the VENDORNAME directory. For example, `REQUIREDVENDORFILES=HP_ADDF.DLL`.

- **OPTIONALVENDORFILES**

  Names of extra files that are optional, that is, not required for correct operation. For example: `OPTIONALVENDORFILES=PCLHELP.HLP, HP_ADDF.SYM, HP_ADDF.MAP`.

- **CLASSNAME (Optional)**

  Name of the output stream created by the hardcopy driver. For example, `CLASSNAME=PCL`.

- **REQUIREDCLASSFILES**

  (Optional unless the driver requires other files) Names of extra files required for correct operation by the driver files in the CLASSNAME directory. For example, `REQUIREDCLASSFILES=GENERIC.DLL`.

- **OPTIONALCLASSFILES**

  Names of extra files that are optional, that is, not required for correct operation. For example: `OPTIONALCLASSFILES=*.FNT, GENERIC.SYM, GENERIC.MAP`.

- **REQUIREDDRIVERFILES**

  Names of extra files required by the driver for correct operation. For example, `REQUIREDDRIVERFILES=LASERJET.DRV`.

- **OPTIONALDRIVERFILES**

  Names of extra files that are optional, that is, not required for correct operation. For example, `OPTIONALDRIVERFILES=LASERJET.SYM, LASERJET.MAP`.

The subdirectory structure on the boot driver created by the hardcopy driver install is as follows:

`OS2\DLL\VENDORNAME\CLASSNAME\DRVNAME` For example: `OS2\DLL\HP\PCL\LASERJET`

Most hardcopy drivers need to be installed only in one subdirectory. In this case, omit CLASSNAME and ensure that VENDORNAME is equal to the name of the DRV file.

** LaserJet is a trademark of the Hewlett-Packard Company

For example:

```
VENDORNAME = PSCRIPT
REQUIREDDRIVERFILES = PSCRIPT.DRV
OPTIONALDRIVERFILES = PSCRIPT.HLP
```

This will install the PostScript** driver into a subdirectory named PSCRIPT.

Version Numbering: The extended attribute shown below can be used for version numbering:

- **.VERSION (Optional)**

  Can be used to indicate the version number of a hardcopy driver, for example, `.VERSION=13.328`. Hardcopy drivers can load this attribute and display it in OS2_PM_DRV_DEVMODE dialogs. See "Hardcopy Driver Migration" for more information.

Improved Performance of Drivers in the Workplace Shell: The following extended attributes can be used to improve the performance of presentation drivers in the Workplace Shell:

- **.EXPAND (Optional)**

  List the device names supported by a hardcopy driver. Improves performance over use of OS2_PM_DRV_DEVICENAMES (especially from a diskette). Names are zero-terminated strings with NULL at the end. For example: `.EXPAND=HP LaserJet II\0HP LaserJet III\0\0`.

- **.ICON (Optional)**

  Gives the icon definition used by the Workplace Shell to display this hardcopy driver. This extended attribute is automatically created during the build process if the ICON keyword is used with a resource identifier of *1*, or if the DEFAULTICON keyword is used in the RC file for a hardcopy driver. For example: `.ICON 1 LASER.ICO`.

- **.HIDDEN (Optional)**

  Indicates which hardcopy driver is used by by files associated with a printer driver. This attribute is used by the Workplace Shell to *not* display these files in a directory folder. For example: `.HIDDEN=LASERJET.DRV`.

Hardcopy Device Names

If a hardcopy driver supports multiple devices, this is indicated at the API and user level. There are three parts:

- The hardcopy driver implements OS2_PM_DRV_DEVICENAMES.
- The hardcopy driver accepts a device name in the szDeviceName field of pdriv. See "Enable Subfunction 02H — FillPhysicalDeviceBlock" on page 7-8.
- The device name is accepted in OS2_PM_DRV_MODES.

Hardcopy Driver Migration

Hardcopy drivers must be able to work with back-level and forward-level drivers across a network. Notice that there are several possible driver version numbers:

- EA Version. Derived from a build number (for example, 13.160)
- Version Number of Dialog. Derived from the build number. Service representatives can use this number to uniquely identify the build of the hardcopy driver and the fix level.

** Postscript is a trademark of Adobe Systems Incorporated

- INI File Data Version. This number identifies to the driver the version number of the data stored in the OS2SYS.INI file. The numbering system and format is hardcopy driver defined.
- lVersion Number. Given in the *DrivData* structure of job or printer properties. The numbering scheme of this ULONG is hardcopy driver defined, but the following format of lVersion is suggested. It is a number of type FIXED:
  - High WORD. System level in BCD (for example, 0020)
  - Low WORD. Build level in BCD (for example, 0160).

For example, a new driver might use 20.0000. The next minor update might be labeled 20.0001. A major update (for a new operating system release) might be labeled 21.0000.

For forward-level drivers reading back-level driver data, the data must be understood (by using the lVersion number as a guide to its format) and any missing data must be defaulted to the device defaults. For back-level drivers reading forward-level driver data, as much as possible should be read and the rest ignored. This implies that forward-level drivers only add new driver data fields at the end of the data.

Hardcopy Driver Output to File

Hardcopy drivers can print data to a disk file so that the print file can later be sent to the printer without the need for the application. The print file can be used as a file interchange format. *Printing to file* means the hardcopy driver converts the *prespool* input into a device-dependent format and stores the output directly to a file. File system errors must be reported.

Two methods the hardcopy driver implements to allow printing to disk are:

- The system-provided method for printing to disk requires that the hardcopy driver handle a file name as a pszLogAddress on a OD_DIRECT Enable Subfunction 02H – FillPhysicalDeviceBlock.
- In addition to the above method, the hardcopy driver can implement printing to disk by changing the job properties dialog to allow the user to input a fully qualified file name. There are circumstances in which the hardcopy driver must know the format of the required output file. For example, a hardcopy driver can output raw PostScript or Encapsulated PostScript (EPS).

Help

All hardcopy drivers have a Help function, which invokes contextual help. The help should be complete and indexed.

Job Error Dialog

The following push buttons on a message box are presented to an end-user:

RETRY Retry sending print data
ABORT Delete job
IGNORE Cancel dialog.

The hardcopy driver will respond to each of the returns in the following manner:

MBID_RETRY Continue sending data to the output buffer (PrtWrite)

MBID_ABORT Issue a PrtAbort to tell the spooler to delete the current job and set a flag that the job has been aborted, then return from the write thread

MBID_IGNORE Continue sending data to the output buffer (PrtWrite).

The Job Error dialog contains a Help pushbutton and associated help.

Part 3. OS/2 2.0 Presentation Drivers

Chapter 3. Display Drivers

This chapter describes the types of exported entry ports used by OS/2 2.0 display drivers.

Exported Entry Points

The following entry points are exported by the dynamic link library of a display driver:

```
EXPORTS
   MoveCursorBlock     @103
   OS2_PM_DRV_QUERYSCREENRESOLUTIONS   /* Optional */
```

MoveCursorBlock

Display drivers must export an entry point for the MoveCursorBlock table. This table contains information about the display driver's MoveCursor routine (code) and data areas. The table values are checked after the display driver is initialized. This allows the driver to determine the correct values.

```
typedef struct _MCDESCRIPTION { /* mod */
    PVOID    pMoveCursor;
    ULONG    ulCodeLength;
    PVOID    pCursorData;
    ULONG    ulDataLength;
} MCDESCRIPTION;
typedef MCDESCRIPTION FAR* PMCDESCRIPTION;
```

The fields in the typedef structure are described below:

pMoveCursor Flat address of the MoveCursor routine in the display driver
ulCodeLength Length in bytes of the MoveCursor routine
pCursorData Flat address of the data area used by the MoveCursor routine
ulDataLength Length in bytes of the data area used by the MoveCursor routine

This routine support calls from the system timer (at interrupt time). The strategy for the MoveCursor routine is that the pointer is checked and, if necessary, redrawn or excluded at timed intervals. The PMDD.SYS physical device driver creates a privilege level 0 alias for the data address and passes the alias to the routine in the EAX register when MoveCursor is called at interrupt time. Therefore, all data addressing within the routine must be performed relative to this address.

At entry to the MoveCursor routine, the stack contains the following:

```
VOID MoveCursor(LONG abs_x, LONG abs_y, PVOID pCursorData)
```

Using the C calling convention, the stack contains two LONGs, which hold the x- and y-coordinates of the cursor hot spot, and a PVOID that is a pointer to the cursor data area that is valid in the current context. All references to this data area must be done relative to the address passed in. If the x- and y-coordinates are set to *0x80000000*, this signifies a CheckCursor call.

Regular timer interrupts give the presentation driver an opportunity to check whether the pointer is valid. For example:

- Have new x- and y- coordinates been set?
- Is the pointer excluded because of a drawing operation. If so, has that operation been completed?
- Is the pointer currently visible (although it should be excluded) because it is in an area that is being redrawn?

At the end of the MoveCursor routine, a check is performed to see if a new location was given for the pointer while it was being drawn. If the pointer has moved again, it must be drawn at the new location or

be excluded because it has moved into the protection rectangle. This implies that the routine needs to track both real (x, y) and pointer (x, y).

Programming Considerations: Typical cursors are an arrow, or a cross, with an action point called the *hot spot* at the point of the arrow or the center of the cross. When the presentation driver draws a cursor, the origin of the image must be offset to place the action point at the required (x, y) position. The required offset is specified in the call to GreSetCursor. Because the cursor entry point can be called at various times from many different places, the cursor routine uses semaphores to protect itself (protection is the responsibility of the presentation driver). Similarly, because cursor drawing can be a time-consuming operation, the display driver must also protect itself against re-entrance.

The display driver must resolve all interactions between cursor drawing at interrupt time and access to video hardware. While in the background, the display driver does not draw any cursor image.

Caution should be used when the display is a buffered device and the cursor is drawn into a bit map in the buffer. In this case, the display driver deletes the cursor and excludes it when a draw operation occurs at the cursor location. To do this, the driver does a *hit test* for each output operation to see if the cursor location is in the drawing area, and to set a protection rectangle that is used to exclude the cursor.

OS2_PM_DRV_QUERYSCREENRESOLUTIONS

OS2_PM_DRV_QUERYSCREENRESOLUTIONS should be exported by display drivers that support multiple display resolutions. This entry point allows the operating system to determine the display modes supported by the display driver.

Entry to the routine is as follows:

```
ULONG QueryResolutions (pBuf, pcbBuf)
```

Parameter	Data Type	Description
pBuf	PVOID	Pointer to the output buffer. See below.
pcbBuf	PULONG	Pointer to the number of bytes in the buffer. See below.

pBuf Pointer to the buffer that receives the output from this function. The output is returned in a SCREENRESCOUNT structure followed by an array of SCREENRESOLUTION structures. This function should fill pBuf with the following information:

```
typedef struct _SCREENRESCOUNT {
   ULONG maxcount;
   ULONG count;
   ULONG res_struct_length;
} SCREENRESCOUNT;
typedef SCREENRESCOUNT *PSCREENRESCOUNT;
```

The SCREENRESCOUNT fields and descriptions are as follows:

maxcount Total number of screen resolutions supported.

count Number of SCREENRESOLUTION structures returned. This count will be less than maxcount if the size of the pBuf buffer was defined too small as identified by pcbBuf.

res_struct_length Length of a SCREENRESOLUTION structure. This value should be used to increment between structures so that any future increase in the size of the structure (to add additional information) will not cause a failure.

The SCREENRESCOUNT information should be followed by the SCREENRESOLUTION information:

```
typedef struct _SCREENRESOLUTION {
   ULONG width;
   ULONG height;
   ULONG colors;
   ULONG planes;
   ULONG floptions
} SCREENRESOLUTION;
typedef SCREENRESOLUTION *PSCREENRESOLUTION;
```

The SCREENRESOLUTION fields and descriptions are as follows:

width Width of the device in pels.

height Height of the device in pels.

colors Number of colors supported in this mode.

planes Number of display planes in this mode.

floptions Identifies optional information for each resolution. Valid values include:

DSP_RESOLUTION_OBTAINABLE	Obtainable with the hardware configuration
DSP_RESOLUTION_DEFAULT	Default resolution

pcbBuf Points to the number of bytes in the pScreenResolution buffer. If pScreenResolution is *0* on input, this function stores the size (in bytes) needed to retrieve all of the screen resolution data in pcbScreenResolution. If not *0*, this field contains the number of bytes actually returned.

Return Codes: The return value depends upon the input value of pcbBuf.

0	Returns the size in bytes needed to retrieve all of the screen resolution data
Non-zero	Returns the number of bytes actually returned in pBuf

Chapter 4. Graphics Engine Hardcopy Drivers

For the Presentation Manager interface, hardcopy devices such as printers and plotters are queued devices. When an application writes to one of these devices, the presentation driver creates a spool file and writes the data to that file. The data is printed when it is complete and the required device is free.

Two instances of a device context are required to support queued data. The first instance is opened as an OD_QUEUED device by the application program. This DC buffers the data, does any processing that is required, and then writes the data to a spool file. The second instance is opened as an OD_DIRECT device by the queue processor. This DC receives data from the spool file, does any processing that is required, and by using the Prt*xxx* interface, sends the data to the physical device driver.

When a device context is opened, the data type given (PM_Q_STD or PM_Q_RAW) is applicable only for the OD_QUEUED device context type.

Exported Entry Points

The following entry points must be exported by a hardcopy driver dynamic link library:

```
EXPORTS
   OS2_PM_DRV_DEVMODE
   OS2_PM_DRV_DEVICENAMES
   DrvInstall                    /* Optional */
   DrvRemove                     /* Optional */

   DialogProc    @2
```

Note: DialogProc is exported by ordinal. The entry point is used by the Presentation Manager interface to manage the dialog initiated by the OS2_PM_DEVMOVE routine (see "OS2_PM_DRV_DEVMODE" on page 4-2).

OS2_PM_DRV_DEVMODE

This handling routine must be compiled to run at Ring 2 Conforming (privilege level 2) or Ring 3 (privilege level 3). The Device Modes entry point is exported by hardcopy drivers as OS2_PM_DRV_DEVMODE to support the DevPostDeviceModes function at the Application Programming Interface (API). In the hardcopy driver, the handling routine generates a DRIVDATA structure that defines the current setting of printer properties or job properties, which identify the options that are set when the job is printed (see "Remarks" on page 4-3). All hardcopy drivers must contain a handling routine for OS2_PM_DRV_DEVMODE.

Applications such as the Presentation Manager Print Object call DevPostDeviceModes to configure the device. Notice that such applications usually call this function twice, first with a NULL value for pDriverData to query the length of the driver's DRIVDATA structure, and then with a valid pointer to get the data.

The syntax used by the Presentation Manager interface to call the Device Modes routine in the hardcopy driver is as follows:

```
LONG APIENTRY OS2_PM_DRV_DEVMODES (pDriverData, pszDriverName, pszDeviceName, pszPrinterName, lOption)

PDRIVDATA   pDriverData;
PSZ         pszDriverName;
PSZ         pszDeviceName;
PSZ         pszPrinterName;
ULONG       lOption;
```

Note: LONG, APIENTRY, PDRIVDATA (DRIVDATA *), and PSZ (char *) are defined in OS2DEF.H, which is included through the header file OS2.H.

Stack Frame: At entry to the device modes routine, the stack frame contains:

Parameter	Description
pDriverData	NULL or pointer to DRIVDATA structure. See below.
pszDriverName	Pointer to a string containing the name of the hardcopy driver.
pszDeviceName	Pointer to a string containing the device name, for example, HP LASERJET II P, as defined by the presentation driver.
pszPrinterName	NULL or pointer to a string containing the printer name, such as PRINTER1, as defined by the user through the Presentation Manager control panel. A NULL pointer or NULL string are both valid conditions for this parameter.
lOption	Option flag. See below.

pDriverData NULL or pointer to memory location for DRIVDATA structure:

- **cb** Number of bytes in the structure.
- **lVersion** Version number of the presentation driver. Subsequently used by the presentation driver to verify its entry in the INI file.
- **szDeviceName[32]** Device name.
- **abGeneralData** Driver-specific data for job or printer properties. See "Remarks" on page 4-3.

If pDriverData is NULL, the handling routine in the presentation driver must return the length of the driver's DRIVDATA structure for either job or printer properties.

lOption Identifies the action that should be taken by the presentation driver:

DPDM_POSTJOBPROP	Display a dialog for job properties and return the DRIVDATA structure. Do not update OS2SYS.INI. The abGeneralData field contains the job properties.
DPDM_CHANGEPROP	Display a dialog for printer properties and device defaults, update the OS2SYS.INI file, and return the DRIVDATA structure. The abGeneralData field contains the printer properties.
DPDM_QUERYJOBPROP	Return the DRIVDATA structure with device default job properties. Do not display a dialog. The abGeneralData field contains the device default job properties.

All other values are reserved.

Return Codes: The handling routine in the presentation driver returns a LONG integer. Valid values are:

DPDM_ERROR	Error
DPDM_NONE	There are no settable options.
>0	The number of bytes for the required DRIVDATA structure when the input pointer pDriverData = NULL.
DEV_OK	The data block pointed to by the input parameter pDriverData was initialized (when the pDriverData input pointer is not equal to NULL).

Remarks: The details about printer properties and job properties are stored as a set of flags or values in the array abGeneralData. Do not store pointers in this array because they might not be valid when they return. This array is driver-specific. What flags are needed and where those flags are in the array must be determined to fully exploit the capabilities of the device.

A list of related terms and their definitions follows.

Properties: This is a descriptive term for software and hardware characteristics of hardcopy (printing, plotting, camera, etc.) devices. For a particular property, there is a list of possible values from which the user can select one or more values. This list can be extended by adding user-defined values but can never be reduced by removing predefined values. For example, predefined forms, Letter and A4, can never be deleted. However, the user can add a form named 'Blue Letter' to describe colored separator paper.

Job Properties: These are properties that can be changed from job to job. Typically, job property values are set in the printer by sending software commands. Some job properties can be derived from printer properties. Examples of typical job properties are:

- Paper orientation
- Forms required
- Device resolution
- Single-sided or duplex printing.

Printer Properties: These are properties that describe the printer physical characteristics. Printer properties are mutually exclusive of job properties (see following description). Examples of typical printer properties are:

- Number of paper bins
- Form size in each paper bin
- Hardware fonts installed
- Font cartridges installed (cartridge slots download fonts at IPL time)
- Availability of optional add-on hardware, such as an envelope feeder or SCSI hard disk available.

User-Definable Values: For each individual property, there is a list of possible values which can be extended by making use of user-definable values (for example, user-defined forms.) User-definable values can only be defined or changed in the printer property dialogs, that is, when OS2_PM_DRV_DEVMODE is called with the DPDM_CHANGEPROP flag.

Selected Values: Given a particular property and its set of values, the user can choose one or more to be *selected* values. For example, Paper Bin 1 contains Letter paper, and the envelope bin accepts Com10, DL, or B5 envelopes.

Device Default Properties: Device defaults can be set in the printer properties dialog. These defaults do not usually contain user-defined values. The selectable values are chosen according to country code and the most common delivery configuration of the device. In a hardcopy driver that supports multiple devices, it is possible that the device default properties are different for different devices.

Option Flags

- DPDM_POSTJOBPROP
- DPDM_CHANGEPROP
- DPDM_QUERYJOBPROP

DPDM_POSTJOBPROP: Any application can call the OS2_PM_DRV_DEVMODE handling routine by using the flag option, DPDM_POSTJOBPROP. The calling program requires the user to select properties for a specific job (draft or letter quality; size, style, and color of the default font, and so forth). The source for the default value of the properties and device defaults is determined by the pszPrinterName parameter:

- If pszPrinterName points to a valid string, the handling routine searches the PM_SPOOLER_DD section of OS2SYS.INI for the abGeneralData associated with the name.
- If pszPrinterName is NULL or points to a NULL string, the handling routine uses the values from abGeneralData in the DRIVDATA structure addressed by pDriverData.

In both cases, the initial default values are used if the handling routine cannot find a valid abGeneralData array.

When called with the DPDM_POSTJOBPROP flag option, hardcopy drivers:

- Examine the pDriverData parameter. If the pDriverData is valid, use these values as a basis of the values to be displayed in the user dialog. If NULL, the driver version number is *0*, or the pDriverData is nonsense, use a set of device default job properties. There are two cases:
  - If the printer name is given, use the job properties.
  - If the printer name is NULL or not a valid printer name, use the job properties derived from the device defaults.
- Update the job property values before displaying the user dialog. If the printer name is given, the printer properties stored by that printer are examined and compared with the job properties. There are two cases:
  - Addition of extra values. If extra printer property values are defined, those that are applicable are added to the user dialog. For example, if a new user-defined form was added to the printer properties, it appears on the list of selectable forms.
  - Removal of values. If printer property values are deleted, the deleted values do not appear in the user dialog. If all the selected values are deleted from the printer properties, the device default becomes the selected value. For example, if a user-defined form is removed from the printer properties, it is replaced by a device default such as Letter (for the United States) or A4 (for France).
- Display the user dialog (the user makes selections). Notice that if the user presses Cancel on the dialog, the hardcopy driver returns pDriverData unmodified. If pDriverData contains zeros or is not understood by the hardcopy driver, the driver should return the default job properties for the device.

- Return the data to the caller. Any updates in the user dialog must be reflected in an updated pDriverData parameter, which is passed back to the caller of OS2_PM_DRV_DEVMODE. This update must occur regardless of whether an actual printer name or NULL printer name was passed in.

DPDM_CHANGEPROP: Only the Workplace Shell calls DevPostDeviceModes using this flag option. Applications that must change options for a particular job use DPDM_POSTJOBPROP.

The calling program requires the user to identify the current settings of the device defaults and select the device default properties. This usually requires two dialogs, one to identify options (such as the paper size currently in the device and details of any memory or font cartridges that are installed), and the other to establish a set of device default properties.

An error is returned if a NULL printer name is used. The pDriverData parameter is ignored. Given that a printer name is passed in, the hardcopy driver needs to retrieve the printer properties for that printer and the individual device for that driver. This means the hardcopy driver must store these properties in a separate place from properties for other printers in order to avoid properties and selected values of one printer interacting with another. An example of printer property interaction might be if one printer has a user-defined form in Bin 1, and the other has no extra forms. It would be possible to send a job to the second printer which uses a form that is only available in the first printer.

The hardcopy driver builds a complex keyname containing the printer name, hardcopy driver and device name. It is recommended that the printer properties be stored under individual property keynames.

This format is easily extended for new printer properties. The format of the appname is:

```
PM_DD_<printer name>,<hardcopy driver>.<device name>
```

For example:

appname = PM_DD_PRINTER1, LASERJET.HP LaserJet II
keyname = BINFORMS
value = Assignment of forms to paper bins
keyname = FORMSDATA
value = Definition of forms

appname = PM_DD_PRINTER2, LASERJET.HP LaserJet II
keyname = BINFORMS
value = Assignment of forms to paper bins.

The advantage of this method is that the appname is standardized, therefore:

- When a printer is renamed, the Workplace Shell can automatically move the data to a new appname without the hardcopy driver involved.
- The initialization file can be cleaned up by the Workplace Shell when a printer is deleted, or when the default hardcopy driver for a printer is changed.

DPDM_QUERYJOBPROP: Any application can call OS2_PM_DRV_DEVMODE by using the DPDM_QUERYJOBPROP flag option to find out the device default job properties. These defaults are derived from the printer properties. There are two cases:

- The printer name is given so the properties can be retrieved from OS2SYS.INI under the PM_DD_Printer*xxx* application name.
- The printer name is not given and the hardcopy driver uses the device default printer properties for that device.

Note: Information concerning the design of dialogs and menus is given in *Common User Access Interface Design Guide*.

OS2_PM_DRV_DEVICENAMES

This handling routine must be compiled to run at Ring 3 (privilege level 3). The device names entry point is exported as OS2_PM_DRV_DEVICENAMES by the presentation driver to support the DevQueryDeviceNames function at the API. Applications such as the Presentation Manager Print Object call DevQueryDeviceNames to determine the device names and descriptions and the data types that the presentation driver supports. Hardcopy drivers must contain a handling routine for OS2_PM_DRV_DEVICENAMES.

Applications usually call this function twice, first with a NULL value for cNames and cDataTypes to query the number of names and data types. After allocating the arrays, the application then calls this function with valid values to get the data. If the value of cNames is NULL at the location addressed by pcNames, the handling routine must update cNames to the actual count of names. If cNames has a valid value, the routine must write device names and device descriptions into the arrays addressed by paDeviceName and paDeviceDesc. Similarly, for cDataTypes, the handling routine either writes a valid value into cDataTypes or writes data-type names into the array addressed by paDataType. Notice that when writing to an array, the routine does not write past the end of the array as defined by the associated count.

The syntax used by the Presentation Manager interface to call the device names routine in the presentation driver is as follows:

```
LONG APIENTRY OS2_PM_DRV_DEVICENAMES (pszDriverName, pcNames, paDeviceName, paDeviceDesc, pcDataTypes,
                                      paDataType, lReserved1, lReserved2)
```

Stack Frame: At entry to the device names function, the stack frame contains:

Parameter	Data Type	Description
pszDriverName	PSZ	Pointer to a string containing the name of the device driver, for example, LaserJet
pcNames	PLONG	Pointer to count of fields, cNames, in DeviceName and DeviceDesc arrays
paDeviceName	PSTR32	Pointer to DeviceName array, char[cNames,32]. Device names are NULL-terminated strings such as 'HP LaserJet II'.
paDeviceDesc	PSTR64	Pointer to DeviceDesc array, char[cNames,64]. Device descriptions are NULL-terminated strings such as 'HP LaserJet II'.
pcDataTypes	PLONG	Pointer to count of fields, cDataTypes in DataType array
paDataType	PSZ	Pointer to DataType array, char[cDataTypes,16]
lReserved1	ULONG	Reserved
lReserved2	ULONG	Reserved

Note: LONG, APIENTRY, and PSZ (char *) are defined in file OS2DEF.H. PSTR16, PSTR32, and PSTR64 are defined as pointers to fixed-length character arrays and are included in the header file OS2.H.

Return Codes: The handling routine in the presentation driver returns a LONG integer. Valid values are:

—1 Successful.
0 Error.

Note: The system expects the successful and error return codes from OS2_PM_DRV_DEVICENAMES to be the opposite of those from OS2_PM_DRV_DEVMODE and the Enable subfunctions.

DrvInstall

Note: This entry point is optional.

This entry point informs the hardcopy driver that it is about to be installed or reinstalled. The driver is given the opportunity to update data in the INI file.

```
void DrvInstall()
```

It is the responsibility of the caller to install the complete multi-file driver by using extended attributes.

DrvRemove

Note: This entry point is optional.

This entry point informs the hardcopy driver that it is about to be removed from the system (deleted from hard disk). The driver is given the opportunity to remove data it owns from the INI file. The hardcopy driver does not use this entry point to delete any of its own datafiles unless they are created after installation.

```
void DrvRemove()
```

It is the responsibility of the caller to remove the complete multi-file driver by using extended attributes.

File System Emulation

Presentation drivers for hardcopy devices use an internal interface to communicate with the device. Hardcopy drivers do not differentiate between different types of ports (for example, LPT1 verses COM1). The Prt*xxx* API routes the data to the appropriate physical device driver. This API also handles semaphoring the port so that two threads do not intermix output.

The internal interface is based on the DOS file system calls DosOpen, DosClose, DosWrite, and so forth. Presentation drivers open a device and receive a handle that identifies the device as a file. Subsequent operations such as writing to the device are implemented by writing to the returned handle.

The following functions are used by the presentation driver:

- PrtAbort
- PrtClose
- PrtDevIOCtl
- PrtOpen
- PrtWrite.

PrtAbort

```
VOID APIENTRY PrtAbort (hDevice)
```

This function aborts operations to the output device identified by hDevice file handle. Any output data that is held in buffers for the physical device driver is emptied. PrtAbort does not close the device, therefore, the presentation driver must call PrtClose after aborting operations. If PrtWrite is called to write to a device whose output has been aborted, the call is not honored by the system.

Parameters

Parameter	Data Type	Description
hDevice	HFILE	Device handle

Return Codes: None. PrtAbort is a VOID function.

Remarks: Presentation drivers do not use PrtClose to abort an output operation. The effect of PrtClose is to output any buffered data before closing the device.

PrtClose

```
ULONG APIENTRY PrtClose (hDevice)
```

This function closes the output device identified by hDevice.

Parameters

Parameter	Data Type	Description
hDevice	HFILE	Device handle

Return Codes: This function returns the same codes as DosClose.

ERROR_ACCESS_DENIED
ERROR_FILE_NOT_FOUND
ERROR_INVALID_HANDLE
NO_ERROR.

Remarks: If this function returns an error, it is reissued to close the device.

PrtDevIOCtl

```
ULONG APIENTRY PrtDevIOCtl (pData, pParms, ulFunction, ulCategory, hDevice)
```

This function passes device-specific commands to the device. PrtDevIOCtl is an emulation of DosDevIOCtl. For a full description of the parameters, see DosDevIOCtl in the *OS/2 2.0 Presentation Manager Programming Reference*. For further information about the IOCtl interface, see the chapter on generic IOCtl commands in the *OS/2 2.0 Physical Device Driver Reference*.

Parameters

Parameter	Data Type	Description
pData	PVOID	Pointer to a data packet
pParms	PVOID	Parameter list
ulFunction	ULONG	Function number
ulCategory	ULONG	Category code
hDevice	HFILE	Device handle

Return Codes: This function returns the same codes as DosDevIOCtl.

ERROR_BAD_DRIVER_LEVEL
ERROR_GEN_FAILURE
ERROR_INVALID_CATEGORY
ERROR_INVALID_DRIVE
ERROR_INVALID_FUNCTION
ERROR_INVALID_HANDLE
ERROR_INVALID_PARAMETER
ERROR_MONITORS_NOT_SUPPORTED
ERROR_PROTECTION_VIOLATION
ERROR_UNCERTAIN_MEDIA
NO_ERROR.

PrtOpen

```
ULONG APIENTRY PrtOpen  (pszDeviceName, phDevice, psAction, cbFile, ulFat , fnOpen, flMode, lRes)
```

This function opens a device file for output and returns its handle in the location addressed by phDevice. If an attempt is made to open a device file that is already open, an error is returned. PrtOpen is an emulation of DosOpen. For a full description of the parameters, see DosOpen in the *OS/2 2.0 Presentation Manager Programming Reference*.

Parameters

Parameter	Data Type	Description
pszDeviceName	PSZ	Pointer to a string identifying the device
phDevice	PHFILE	Pointer to a location for the returned file handle for the device
psAction	PUSHORT	Pointer to location for the returned value that identifies the action taken by the system
cbFile	ULONG	Initial size, in bytes, of the output file
ulFat	ULONG	File attribute
fnOpen	ULONG	Open function type
flMode	ULONG	Open mode of file
lRes	ULONG	Reserved. Must be *0*.

Return Codes: This function returns the same codes as DosOpen.

ERROR_ACCESS_DENIED
ERROR_CANNOT_MAKE
ERROR_DEVICE_IN_USE
ERROR_DISK_FULL
ERROR_DRIVE_LOCKED
ERROR_FILE_NOT_FOUND
ERROR_FILENAME_EXCED_RANGE
ERROR_INVALID_ACCESS
ERROR_INVALID_PARAMETER
ERROR_NOT_DOS_DISK
ERROR_OPEN_FAILED
ERROR_PATH_NOT_FOUND
ERROR_PIPE_BUSY
ERROR_SHARING_BUFFER_EXCEEDED
ERROR_SHARING_VIOLATION
ERROR_TOO_MANY_OPEN_FILES
NO_ERROR.

PrtWrite

```
ULONG APIENTRY PrtWrite (hDevice, pvoidData, cData, cWritten)
```

This function writes data to the device file identified by hDevice.

Parameters

Parameter	Data Type	Description
hDevice	HFILE	Device handle
pvoidData	PVOID	Pointer to data buffer
cData	ULONG	Length of data in bytes
cWritten	PULONG	Pointer to a count of bytes actually written to the device

Return Codes: This function returns the same codes as DosWrite.

ERROR_ACCESS_DENIED
ERROR_BAD_UNIT
ERROR_BROKEN_PIPE
ERROR_INVALID_HANDLE
ERROR_LOCK_VIOLATION
ERROR_NOT_DOS_DISK
ERROR_OUT_OF_PAPER
ERROR_WRITEFAULT
NO_ERROR.

Remarks: Some *physical device drivers* return NO_ERROR even though cWritten is not equal to cData. Presentation drivers should compare cData to cWritten to determine if a request has completed successfully before checking for return codes. To complete the request when cData is not equal to cWritten, the call must be issued after calculating the new starting point (pvoidData = pvoidData+cWritten) and the remaining characters to transfer (cData = cData–cWritten).

Spooler Components

The spooler interface (Spl...) is implemented in two libraries, PMSPL.DLL and PMPRINT.QPR, that support four activities:

- Management of spool buffers for PM_Q_STD data (SplStd*xxx* interface)
- Management of queued files (SplQm*xxx* interface)
- Processing of queued files (SplQp*xxx* interface)
- Displaying messages on the screen (SplMesageBox).

Spool File Creation

The steps taken by the presentation driver to create a spool file are determined by the data type for which the DC was enabled. All presentation drivers must support PM_Q_STD and PM_Q_RAW (overviews of creating a spool file for these data types are shown below).

Data types can be defined by the user. A name should be chosen that is not likely to clash with other user-defined data types. The name must be a string of up to 16 characters in the range of *A*—*Z*, *0*—*9*, or _. Notice that the data type is only useful to applications that know about it, and with presentation drivers that implement it.

Print jobs must be spooled by using the data type given on the call to Enable Subfunction 02H — FillPhysicalDeviceBlock. Therefore, print jobs queued PM_Q_STD can be printed with queue processor options applied to the job.

PM_Q_STD

PM_Q_STD data is a file of Gpi*xxx* calls that describe the output document. This data type is independent of the device and the presentation driver. The presentation driver invokes the spooler's standard interface (SplStd*xxx*). All subsequent Gpi*xxx* calls and some escape codes sent to the DC are recorded in a spool buffer. When DEVESC_ENDDOC is detected, the presentation driver ends the recording and invokes the spooler's interface (SplQm*xxx*) to write the buffered data into a spool file.

Normal sequence of events:

1. Application calls DevOpenDC to open an OD_QUEUED device for printing PM_Q_STD data.
   a. Hardcopy driver calls SplStdOpen to open a recording.
2. Application calls DevEscape with DEVESC_STARTDOC.
   a. Hardcopy driver calls SplStdStart to start recording.
   b. Spooler records all Gpi*xxx* calls and some escape codes in the spool buffer.

      Note: Recording does not stop the flow of Gpi*xxx* calls through the system. These calls are processed and the resulting Gre*xxx* calls are passed on to handling routines in the graphics engine and presentation driver. Special considerations apply for escape codes. For details, see the individual escape codes under "GreEscape" on page 8-15.

   c. Hardcopy driver DC is OD_INFO. The presentation driver tracks the current position, does any bounds calculation required, and responds to queries from the application. In particular, the presentation driver must be able to understand and reply to:
      - DevQueryCaps
      - DevQueryHardCopyCaps
      - DevQueryDeviceNames
      - DevPostDeviceModes.
3. Application calls DevEscape with DEVESC_NEWFRAME.

a. Hardcopy driver resets current position and bounds.

4. Application calls DevEscape with DEVESC_ENDDOC.

   a. Hardcopy driver calls SplStdStop to stop the recording.

   b. Hardcopy driver calls SplQmOpen and SplQmStartDoc to open and start a spool file.

   c. Hardcopy driver calls SplStdQueryLength to get the length, in bytes, of the spooled data.

   d. Hardcopy driver calls SplStdGetBits to get data from the spool buffer into memory that is owned by the presentation driver. (The driver might need to loop on this step and the next if the spooled data is larger than the available memory.)

   e. Hardcopy driver calls SplQmWrite to write the data in the spool file.

   f. Hardcopy driver calls SplStdDelete to delete the data in the spool buffer.

   g. Hardcopy driver calls SplQmEndDoc to stop the spool file, and returns the Job ID to the application's DevEscape with DEVESC_ENDDOC.

5. Application calls DevEscape with DEVESC_STARTDOC (repeat Step 2).

6. Application calls DevEscape with DEVESC_NEWFRAME (repeat Step 3).

7. Application calls DevEscape with DEVESC_ENDDOC to spool the second job (repeat Step 4).

8. Application calls DevCloseDC.

   a. Hardcopy driver calls SplStdClose and SplQmClose to close the spool buffer and the spool file.

Abort sequence of events:

1. Application calls DevOpenDC to open an OD_QUEUED device for printing PM_Q_STD data.

   a. Hardcopy driver calls SplStdOpen to open a recording.

2. Application calls DevEscape with DEVESC_STARTDOC.

   a. Hardcopy driver calls SplStdStart to start recording.
   b. Spooler records all Gpi*xxx* calls and some escape codes in the spool buffer.

      Note: Recording does not stop the flow of Gpi*xxx* calls through the system. These calls are processed and the resulting Gre*xxx* calls are passed on to handling routines in the graphics engine and presentation driver. Special considerations apply for escape codes. For details, see the individual escape codes under "GreEscape" on page 8-15.

   c. Hardcopy driver behaves as if the DC was opened as OD_INFO. The presentation driver tracks the current position, does any bounds calculation required and responds to queries from the application. In particular, the presentation driver must be able to understand and reply to:
      - DevQueryCaps
      - DevQueryHardCopyCaps
      - DevQueryDeviceNames
      - DevPostDeviceModes.

   The calls can be journaled but the journal file is not saved.

3. Application calls DevEscape with DEVESC_ABORTDOC to abort the document.

   a. Hardcopy driver calls GreStopJournalFile.
   b. Hardcopy driver calls GreDeleteJournalFile.
   c. Hardcopy driver calls SplStdStop to stop the recording.
   d. Hardcopy driver calls SplStdDelete to delete the data in the spool buffer.
   e. Hardcopy driver calls SplQmAbortDoc to stop the spool file.

4. The application calls DevCloseDC.

   a. Hardcopy driver calls SplStdClose and SplQmClose to close the recording and the spool file.

PM_Q_RAW

PM_Q_RAW data is a device-dependent bit stream.

Normal sequence of events:

1. Application calls DevOpenDC to open an OD_QUEUED device for printing PM_Q_RAW data.
   a. Hardcopy driver calls SplQmOpen to open a spool file.
2. Application calls DevEscape with DEVESC_STARTDOC.
   a. Hardcopy driver calls GreCreateJournalFile to create a journal file.
   b. Hardcopy driver calls SplQmStartDoc to start the spool file.
   c. Hardcopy driver calls GreStartJournalFile to start recording Gre*xxx* calls.

      Note: Recording does not block the flow of Gre*xxx* calls to handling routines in the graphics engine and presentation driver.

   d. Hardcopy driver processes the incoming Gre*xxx* calls to create the first band of data for the spool file.
3. Application calls DevEscape with DEVESC_NEWFRAME to start new page.
   a. Hardcopy driver calls GreStopJournalFile to stop the journal file.
   b. Hardcopy driver calls SplQmWrite to write the first band in the spool file.
   c. Hardcopy driver moves clip rectangle to next band in presentation space.
   d. Hardcopy driver calls GrePlayJournalFile to play the journal file.
   e. Hardcopy driver processes each Gre*xxx* call to create the second band of data for the spool file.
   f. Hardcopy driver calls SplQmWrite to write the second band in the spool file.
   g. Hardcopy driver plays the journal file repeatedly until all the bands have been processed and passed to the spooler.
   h. Hardcopy driver issues a page eject, if necessary.
   i. Hardcopy driver calls GreDeleteJournalFile.
   j. Hardcopy driver calls SplQmEndDoc to stop the spool file, and returns the Job ID to the application's DevEscape with DEVESC_ENDDOC.

   Note: If no Gre*xxx* calls have been made (that is, no output is required), only Steps a and h are executed.
4. Application calls DevEscape with DEVESC_ENDDOC to end the document.
   a. Hardcopy driver calls GreStopJournalFile to stop the journal file.
   b. Hardcopy driver calls SplQmWrite to write the first band in the spool file.
   c. Hardcopy driver calls GrePlayJournalFile to play the journal file.
   d. Hardcopy driver processes each Gre*xxx* call to create the second band of data for the spool file.
   e. Hardcopy driver calls SplQmWrite to write the second band in the spool file.
   f. Hardcopy driver plays the journal file repeatedly until all the bands have been processed and passed to the spooler.
   g. Hardcopy driver calls GreDeleteJournalFile.
   h. Hardcopy driver calls SplQmEndDoc to stop the spool file, and returns the Job ID to the application's DevEscape with DEVESC_ENDDOC.

Note: If no Gre*xxx* calls have been made (that is, no output is required), only Steps a, i, and j are executed. No page is ejected.

5. Application calls DevEscape with DEVESC_STARTDOC (repeat Step 2).
6. Application calls DevEscape with DEVESC_NEWFRAME to start new page (repeat Step 3).
7. Application calls DevEscape with DEVESC_ENDDOC to spool the second job (repeat Step 4).
8. Application calls DevCloseDC.
   a. Hardcopy driver calls SplQmClose.

Abort sequence of events:

1. Application calls DevOpenDC to open an OD_QUEUED device for printing PM_Q_RAW data.
   a. Hardcopy driver calls SplStdOpen to open a spool file.
2. Application calls DevEscape with DEVESC_STARTDOC to start the document.
   a. Hardcopy driver calls GreCreateJournalFile to create a journal file.
   b. Hardcopy driver calls SplQmStartDoc to start the spool file.
   c. Hardcopy driver calls GreStartJournalFile to start recording Gre*xxx* calls.

      Note: Recording does not block the flow of Gre*xxx* calls to handling routines in the graphics engine and presentation driver.

   d. Hardcopy driver processes the incoming Gre*xxx* calls to create the first band of data for the spool file.
3. Application calls DevEscape with DEVESC_ABORTDOC to abort the document.
   a. Hardcopy driver calls GreStopJournalFile.
   b. Hardcopy driver calls GreDeleteJournalFile.
   c. Hardcopy driver calls SplQmAbortDoc to stop the spool file.

      Note: The hardcopy driver can be processing the DEVESC_ENDDOC (banding, for example) and a DEVESC_ABORT comes into the hardcopy driver on another thread.

4. Application calls DevCloseDC.
   a. Hardcopy driver calls SplQmClose to close the spool file.

Querying and Setting Configuration Data

The configuration of printers and queues is stored in OS2SYS.INI. It is recommended that the Spl*xxx*Device and Spl*xxx*Queue functions are used as a high-level interface into OS2SYS.INI. Refer to the *OS/2 2.0 Presentation Manager Programming Reference* for further information.

Spooler Support Functions

The purpose of the spooler is to control the queues, create new spool files, and invoke the queue processor when a job is ready for printing. The spooler also provides a function, SplMessageBox, that can be called to display a message to the user.

The following functions are available in the spooler:

- SplMessageBox
- SplQmAbort
- SplQmAbortDoc
- SplQmClose
- SplQmEndDoc
- SplQmOpen
- SplQmStartDoc
- SplQmWrite.

SplMessageBox

```
ULONG APIENTRY SplMessageBox (pszAddress, flErrorInfo, flErrorData, pszText, pszCaption, idWindow, fsStyle)
```

This function creates and displays a message box. SplMessageBox is similar to WinMessageBox. For details, see WinMessageBox in the *OS/2 2.0 Presentation Manager Programming Reference.*

Parameters

Parameters	Data Type	Description
pszAddress	PSZ	Pointer to a string containing the logical address of the device, such as 'LPT1'.
flErrorInfo	ULONG	Error information. See below.
flErrorData	ULONG	Error data. See below.
pszText	PSZ	Pointer to the text string for the message box.
pszCaption	PSZ	Pointer to a string containing a meaningful title for the message box. The text is centered in the title bar. If more than 40 characters are supplied, excess characters at the beginning and end of the string are not displayed.
idWindow	USHORT	Window ID of the message box window.
fsStyle	USHORT	This bit array specifies the contents and function of the message box.

flErrorInfo Error information. One of the following flags must be set to identify where the error occurred:

SPLINFO_QPERROR	Spooler queue processor error
SPLINFO_DDERROR	Presentation driver error
SPLINFO_SPLERROR	Spooler error
SPLINFO_OTHERERROR	Any other error.

One of the following flags is also set to indicate the severity of the error:

SPLINFO_INFORMATION	Information only, no error
SPLINFO_WARNING	Warning
SPLINFO_ERROR	Recoverable error
SPLINFO_SEVERE	Severe, irrecoverable error.
SPLINFO_USERINTREQD	This flag is optional. It shows that recovery requires action from the user.

flErrorData Error data:

SPLDATA_PRINTERJAM	Printer is jammed, offline, or not powered on.
SPLDATA_FORMCHGREQD	Form change required
SPLDATA_CARTCHGREQD	Font cartridge change required
SPLDATA_PENCHGREQD	Pen change required
SPLDATA_DATAERROR	Data error, such as missing file
SPLDATA_UNEXPECTERROR	Unexpected DOS error
SPLDATA_OTHER	Any other error.

Return Codes: This function returns a USHORT value (sResponse) that indicates the user's response.

SplQmAbort

```
BOOL APIENTRY SplQmAbort (hspl)
```

This function aborts and closes the spool file identified by hspl.

Parameters

Parameters	Data Type	Description
hspl	HSPL	Spooler handle

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplQmAbortDoc

```
BOOL APIENTRY SplQmAbortDoc (hspl)
```

This function aborts the document on the spool file identified by hspl. All data for that document, including SplQmStartDoc data, is erased. SplQmAbortDoc does not close the spool file. The presentation driver can restart the document by using the same spooler handle. If the hardcopy driver wants to abort the job and close the file, it calls SplQmAbort.

Parameters

Parameters	Data Type	Description
hspl	HSPL	Spooler handle

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplQmClose

```
BOOL APIENTRY SplQmClose (hspl)
```

This function closes the spool file identified by hspl. SplQmClose corresponds to DevCloseDC.

Parameters

Parameters	Data Type	Description
hspl	HSPL	Spooler handle

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplQmEndDoc

```
ULONG APIENTRY SplQmEndDoc (hspl)
```

This function ends the document on the spool file identified by hspl. SplQmEndDoc corresponds to the DEVESC_ENDDOC escape code. The return code, if not SPL_ERROR, is the spooler's Job ID for the document.

Parameters

Parameters	Data Type	Description
hspl	HSPL	Spooler handle

Return Codes

idJobId Job identifier
SPL_ERROR Error.

SplQmOpen

```
HSPL APIENTRY SplQmOpen (pszToken, cbData, pbData)
```

This function opens the spooler for output to a spool file. SplQmOpen is similar to SplStdOpen. The return code, if not SPL_ERROR, is the handle that identifies the spool file.

Parameters

Parameters	Data Type	Description
pszToken	PSZ	Pointer to token. This is a dummy parameter and is specified as **.
cbData	LONG	Number of elements in the data structure.
pbData	PQMOPENDATA	Pointer to a DEVOPENSTRUC structure. See below.

pbData Pointer to a DEVOPENSTRUC structure containing information from the presentation driver's physical device block (see "Enable Subfunction 02H – FillPhysicalDeviceBlock" on page 7-8):

pszLogAddress Pointer to the name of the queue

pszDriverName Pointer to the name of the presentation device driver

pdriv Pointer to a DRIVDATA structure:

- **cb** Number of bytes in structure
- **lVersion** Version number
- **szDeviceName[32]** Device name
- **abGeneralData** Driver-specific data

pszDataType Pointer to a string defining the data type of the queued file. All queue processors must support PM_Q_STD and PM_Q_RAW.

pszComment Pointer to a natural language description of the file

pszQueueProcName Pointer to the name of the queue processor

pszQueueProcParams Pointer to a string of queue processor parameters

pszSpoolerParams Pointer to a string of spooler parameters separated by one or more blanks. Valid parameters are:

FORM=aaa Identifies the form name for the print job. Multiple names are separated by commas (aaa,bbb,ccc). If this parameter is not present in the string of spooler parameters, the job is printed on the current form.

Form names are defined by the presentation driver. Valid names are those that would be returned from a call to the driver's GreQueryHardcopyCaps handling routine.

PRTY=n Identifies the priority for the print job. The priority can be any value from *1–99* (*1* is lowest priority). If this parameter is not present, the priority value defaults to *50*.

pszNetworkParams Pointer to string of networking parameters. These are only used in a network environment and their nature is defined by the network application.

Return Codes: This function returns the spooler handle (hspl), or SPL_ERROR if an error occurred.

SplQmStartDoc

```
BOOL APIENTRY SplQmStartDoc (hspl, pszDocName)
```

This function signals the start of the document for the spool file and supplies a name that the spooler can use to identify the job to the user. SplQmStartDoc corresponds to DevEscape(DEVESC_STARTDOC).

Parameters

Parameter	Data Type	Description
hspl	HSPL	Spooler handle.
pszDocName	PSZ	Pointer to the document name, which can be displayed by the spooler to the user.

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplQmWrite

```
BOOL APIENTRY SplQmWrite (hspl, cbData, pbData)
```

This function writes data from the presentation driver's buffer to the spool file.

Parameters

Parameter	Data Type	Description
hspl	HSPL	Spooler handle
cbData	LONG	Length of the buffer in bytes
pbData	PBYTE	Pointer to the start of the buffer

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

Remarks: The size of the data buffer must not be greater than 64KB. Print jobs that exceed the maximum buffer size must be written by repeatedly calling to this function.

Spooler Support for PM_Q_STD Data Type

The following functions are available to help the hardcopy driver create a spool file containing PM_Q_STD data:

- SplStdClose
- SplStdDelete
- SplStdGetBits
- SplStdOpen
- SplStdQueryLength
- SplStdStart
- SplStdStop.

The format of the PM_Q_STD data is a Presentation Manager metafile. Refer to the *OS/2 2.0 Presentation Manager Programming Reference* for format detail.

SplStdClose

```
BOOL APIENTRY SplStdClose (hdc)
```

This function closes the PM_Q_STD buffer. The call to SplStdClose is made from the presentation driver's BeginCloseDC routine when the device type is OD_QUEUED and the data type is PM_Q_STD. SplStdClose must not be called at any other time.

Parameters

Parameter	Data Type	Description
hdc	HDC	Device context handle

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplStdDelete

```
BOOL APIENTRY SplStdDelete (hstd)
```

This function deletes the PM_Q_STD buffer identified by hstd. Any data in the buffer is lost.

Parameters

Parameter	Data Type	Description
hstd	HSTD	Handle to PM_Q_STD data

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplStdGetBits

```
BOOL APIENTRY SplStdGetBits (hstd, lStart, cBytes, pAddress)
```

This function transfers data from the identified PM_Q_STD buffer into a buffer owned by the presentation driver. Before calling SplStdGetBits, the presentation driver calls SplStdQueryLength to determine the length of the PM_Q_STD data. Depending upon the length, the hardcopy driver allocates a buffer large enough to contain all of the data, or allocates a smaller buffer and receives the data in a series of calls to SplStdGetBits.

Parameters

Parameter	Data Type	Description
hstd	HSTD	Handle to the PM_Q_STD buffer.
lStart	LONG	Offset to the byte at which transfer must start. Used when the data is obtained in a series of calls to SplStdGetBits.
cBytes	LONG	Number of bytes to transfer.
pAddress	PCH	Pointer to the presentation driver's data buffer.

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplStdOpen

```
BOOL APIENTRY SplStdOpen (hdc)
```

This function opens a spool buffer for PM_Q_STD data. The call to SplStdOpen is made from the hardcopy driver's CompleteOpenDC routine when the device type is OD_QUEUED and the data type is PM_Q_STD. SplStdOpen must not be called at any other time.

Parameters

Parameter	Data Type	Description
hdc	HDC	Device context handle

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplStdQueryLength

```
LONG APIENTRY SplStdQueryLength (hstd)
```

This function returns the number of data bytes in the PM_Q_STD buffer identified by hstd.

Parameters

Parameter	Data Type	Description
hstd	HSTD	Handle to the PM_Q_STD buffer

Return Codes: This function returns the size of hstd (cBytes), or SPL_ERROR if an error occurred.

SplStdStart

```
BOOL APIENTRY SplStdStart (hdc)
```

This function starts the recording of GPI and DevEscape calls as a metafile in the PM_Q_STD buffer. Notice that the calls are still processed by the system and passed on to the graphics engine and hardcopy driver as calls to the relevant Gre*xxx* functions.

The hardcopy driver's Escape routine usually calls SplStdStart when DEVESC_STARTDOC is received. Some prior-version presentation drivers made this call from the CompleteOpenDC routine to accommodate applications that did not call DevEscape(DEVESC_STARTDOC) at the start of the document.

Parameters

Parameter	Data Type	Description
hdc	HDC	Device context handle

Return Codes: This function returns BOOLEAN (fSuccess):

TRUE Successful
FALSE Error.

SplStdStop

```
HSTD APIENTRY SplStdStop (hdc)
```

This function stops the recording of GPI and DevEscape calls in the PM_Q_STD buffer. The hardcopy driver's Escape routine calls SplStdStop when DEVESC_ENDOC is received.

Parameters

Parameter	Data Type	Description
hdc	HDC	Device context handle

Return Codes: If successful, SplStdStop returns the handle to the buffer (hstd) that contains the recorded GPI and DevEscape data. If an error occurs, the function returns SPL_ERROR.

Chapter 5. Queue Drivers (Queue Processors)

The Workplace Shell uses the term *queue driver* to identify the queue processor. Each queue has its own queue processor, which prints a spool file. The Presentation Manager interface delivers two different queue drivers. Presentation Manager system queue drivers are supplied in the files, PMPRINT.QPR and PMPLOT.QPR.

The following functions provide an interface to the queue driver:

- SplQpClose
- SplQpControl
- SplQpInstall
- SplQpOpen
- SplQpPrint
- SplQpQueryDt
- SplQpQueryFlags

The spooler calls these functions by using DosLoadModule or DosGetProcAddr, and expects a 16-bit interface. In addition to these functions, a visual interface is supplied through the SplMessageBox entry point.

The user can supply queue drivers to support user data types, however, any queue driver created by the user must support PM_Q_STD and PM_Q_RAW standard data types.

How a Queue Driver Prints

The method used by the queue driver to write data to the hardcopy device depends on whether the data type is PM_Q_STD or PM_Q_RAW.

PM_Q_STD

PM_Q_STD performs in the following manner:

1. Opens a DC for the hardcopy device by using DevOpenDC and enables it as an OD_DIRECT device
2. Calls DevEscape with DEVESC_STARTDOC
3. Writes data which is in metafile format by using GpiPlayMetafile
4. Calls DevEscape with DEVESC_ENDDOC
5. Closes hardcopy device DC by using DevCloseDC.

PM_Q_RAW

PM_Q_RAW performs in the following manner:

1. Opens a DC for the hardcopy device by using DevOpenDC and enables it as an OD_DIRECT device
2. Calls DevEscape with DEVESC_STARTDOC
3. Writes the data using DevEscape with DEVESC_RAWDATA
4. Calls DevEscape with DEVESC_ENDDOC
5. Closes hardcopy device DC by using DevCloseDC.

User Data Types

The processing required for user data types depends on the format of the data type. In some instances it might be necessary to create a special queue processor to support the data type.

SplQpClose

```
BOOL APIENTRY SplQpClose (hproc)
```

This function closes the queue driver (queue processor).

Parameters

Parameter	Data Type	Description
hproc	HPROC	Queue processor handle

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

SplQpControl

```
BOOL APIENTRY SplQpControl (hproc, cmdCode)
```

This function controls the printing of a document.

Parameters

Parameter	Data Type	Description
hproc	HPROC	Queue processor handle
cmdCode	LONG	Control codes. See below.

cmdCode Control codes are as follows:

SPLC_ABORT Printing is aborted and the queue driver is closed.

SPLC_PAUSE Printing is paused. The spool file must not be open or allocated by the queue driver.

SPLC_CONTINUE Printing resumes on a paused job. The spool file is unaltered.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

SplQpInstall

```
BOOL APIENTRY SplQpInstall (hwnd)
```

This function allows the user to configure a queue driver. SplQpInstall is optional.

Parameters

Parameter	Data Type	Description
hwnd	HWND	Window handle

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Remarks: This function is called by the Workplace Shell. It is used to display a dialog to the user for queue driver (queue processor) configuration. The queue driver then stores the values in the OS2SYS.INI file.

SplQpOpen

```
HPROC APIENTRY SplQpOpen (cbData, pbData)
```

This function opens the queue driver and returns its handle. SplQpOpen is normally called by the spooler.

Parameters

Parameter	Data Type	Description
cbData	LONG	Number of elements in the SQPOPENDATA structure
pbData	PSQPOPENDATA	Pointer to SQPOPENDATA structure. See below.

pbData Pointer to SQPOPENDATA structure. This structure is based on the DEVOPENSTRUC and would typically contain data extracted from the presentation driver's physical device block.

pszLogAddress Pointer to logical address.

pszDriverName Pointer to presentation driver name.

pdriv Pointer to a DRIVDATA structure:

cb Size, in bytes, of this structure.

lVersion Version number of the data. Version numbers are defined by the presentation driver.

szDeviceName[32] String identifying the device. Valid values are supplied by the presentation driver.

abGeneralData General data, the type of which is defined by the presentation driver.

pszDataType Pointer to the data type of the queued file. All queue drivers must support PM_Q_STD and PM_Q_RAW. User-defined data types are optional.

pszComment Pointer to a natural language description of the file which could, for example, be displayed by the spooler to the user.

pszProcParams Pointer to a string of queue driver parameters.

pszSpoolerParams Pointer to a string of spooler parameters separated by one or more blanks. Valid parameters are:

FORM=aaa Identifies the form name for the print job. Multiple names should be separated by commas (aaa,bbb,ccc). If this parameter is not present, the job is printed on the current form.

Form names are defined by the presentation driver. Valid names are those that would be returned from a call to the driver's GreQueryHardcopyCaps handling routine.

PRTY=n Identifies the priority for the print job. The priority can be any value from *1—99* (1 is lowest priority). Notice that this parameter has no significance when passed to SplQpOpen.

pszNetworkParams Pointer to string of networking parameters. These are only used in a network environment and their nature is defined by the network application. Typically, the queue driver ignores this parameter.

pszDocName Pointer to a string containing the document name.

pszQueueName	Pointer to a string containing the name of the queue from which the job was sent by the spooler.
pszToken	Pointer to the device information token. This identifies additional device information held in the initialization file. This information is the same as that which can be pointed to by pbData. Any information obtained from pbData overrides the information obtained by using this parameter. If Token is specified as *, no device information is taken from the initialization file. OS/2 2.0 behaves as if * is specified but it allows any string.
idJobId	This is a USHORT value that identifies the print job.

Return Codes: On completion, the handling routine must return:

SPL_ERROR	Error
≠0	Queue driver (queue processor) handle.

SplQpPrint

```
BOOL APIENTRY SplQpPrint (hproc, pszFilename)
```

This function processes and prints the spool file. The handling routine in the queue driver opens a DC for an OD_DIRECT device type. This instance of a DC processes the spooled data, and by using the Prt*xxx* interface, passes the output through the physical device driver to the physical device. When the print job ends, the queue driver assumes that the device is set at the start of a new page and issues a form feed to the device.

Parameters

Parameter	Data Type	Description
hproc	HPROC	Queue processor handle
pszFilename	PSZ	Pointer to name of file containing the data

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

SplQpQueryDt

```
BOOL APIENTRY SplQpQueryDt (pcDatatypes, papszDatatypes)
```

This function returns a list of supported data types.

Parameters

Parameter	Data Type	Description
pcDatatypes	PLONG	Pointer to a value indicating the maximum number of data types. On return, the value is updated to show the number of data types returned.
papszDatatypes	PSZ	Pointer to an array of pointers that address the locations for the returned data type names. Each location must be an array of 16 characters to accommodate a name of the maximum length with its terminating NULL.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Remarks: This function should be called once with pcDatatypes set to *0* to determine the number of data types. The array papszDatatypes is not updated in this instance. The application can then allocate storage for the array and call the function a second time to return a list of supported data types.

SplQpQueryFlags

```
BOOL APIENTRY SplQpQueryFlags (pulFlags)
```

This (optional) function queries print queue processor flags.

Parameters

Parameter	Data Type	Description
pulFlags	PULONG	Points to ULONG to receive flags value

Return Codes: On completion, this function returns BOOLEAN (fSuccess).

TRUE Successful, that is, *pulFlags=QP_RAWDATA_BYPASS(0x0001)
FALSE Not supported

Remarks: This function is called to determine if this print queue processor allows the spooler to bypass it for PM_Q_RAW jobs. By exporting SplQpQueryFlags and setting the contents of pulFlags to QP_RAWDATA_BYPASS, the spooler can bypass calling this print queue processor to print PM_Q_RAW jobs that are still spooling.

Chapter 6. Port Drivers

Port drivers are dynamic link libraries (DLLs) that contain a set of 32-bit functions which provide helper functions for the spooler and Workplace Shell. For each port driver DLL, there should be a physical Port device driver (SYS file) installed in CONFIG.SYS. The filetype of a port driver is *PDR*. The operating system, by default, provides two port drivers named SERIAL.PDR and PARALLEL.PDR.

LPT4−9 are reserved and are installed into OS2SYS.INI by default. The Workplace Shell uses LPT4−9 for NET USEs to a remote print queue. No port driver is provided by LPT4−9, or for ports that are *named pipes*.

The functions exported from a port driver are:

- SplPdEnumPort
- SplPdGetPortIcon
- SplPdInitPort
- SplPdInstallPort
- SplPdQueryPort
- SplPdRemovePort
- SplPdSetPort
- SplPdTermPort.

OS2SYS.INI File Structure

The contents of the INI file section for ports is as follows:

- Port driver PDR install:

```
appname: PM_PORT_DRIVER  keyname: <name>  value: <full path to port driver>

For example:
PM_PORT_DRIVER, SERIAL,    C:\OS2\DLL\SERIAL.PDR
                PARALLEL,  C:\OS2\DLL\PARALLEL.PDR
                EXTLPT,    C:\OS2\DLL\EXTLPT.PDR
```

  This is written by the Workplace Shell during installation.

- Ports known to the system:

```
appname: PM_<portname>  keyname: DESCRIPTION     value: <port description>
                        keyname: INITIALIZATION  value: <initialization values>
                        keyname: TERMINATION     value: <termination values>
                        keyname: PORTDRIVER      value: <name of port driver>
                        keyname: TIMEOUT         value: <timeout in seconds>
For example:
PM_COM1, DESCRIPTION,        Serial Port COM1;
PM_COM1, INITIALIZATION,     9600;0;W;8;1;
PM_COM1, TERMINATION,        ;
PM_COM1, PORTDRIVER,         SERIAL;
PM_COM1, TIMEOUT,            45;
```

  For compatibility, the existing port structure is also supported by all port drivers.

```
appname: PM_SPOOLER_PORT  keyname: <port>  value: <port init/term string>
For example:
PM_SPOOLER_PORT, LPT1,    ;
                 COM1,    9600;0;W;8;1;
```

  The port driver is expected to maintain its own sections in the OS2SYS.INI file. The format of the values stored under the keynames INITIALIZATION and TERMINATION is port driver specific. Also, the port driver might need to hold extra data under the PM_<port> appname.

SplPdEnumPort

```
APIERR APIENTRY SplPdEnumPort (hab, pBuf, cbBuf, pulReturned, pulTotal, pcbNeeded)
```

This function enumerates the port names and port descriptions that this port driver can manipulate.

Parameters

Parameters	Data Types	Description
hab	HAB	Handle to anchor block
pBuf	PVOID	Pointer to buffer of data structures
cbBuf	ULONG	Size of buffer in bytes
pulReturned	PULONG	Number of port entries returned
pulTotal	PULONG	Total number of port entries
pcbNeeded	PULONG	Size of buffer required for all data

Return Codes: This handling routine returns the following errors.

ERROR_INSUFFICIENT_BUFFER
ERROR_MORE_DATA
NO_ERROR.

The buffer, on return, consists of an array of the following data structure:

```
typedef struct _PORTNAMES {
        PSZ pszPortName;
        PSZ pszPortDesc;
} PORTNAMES;
```

Remarks: If the buffer is too small to hold even one data structure, then the error code, ERROR_INSUFFICIENT_BUFFER is returned. If there is more data, ERROR_MORE_DATA, is returned and cbRequired gives the size of buffer required to get all the data.

The Workplace Shell is the expected caller of this function and determines which ports to install in OS2SYS.INI. The port descriptions are used to label the icon for each port. The Workplace Shell does not allow the installation of ports that can be serviced by more than one port driver. The first port installed is used. The Workplace Shell does not display other ports for installation. If another port is required, the first port must be de-installed.

SplPdGetPortIcon

```
BOOL SplPdGetPortIcon (hab, idIcon)
```

This function queries the Resource ID of the icon that represents the port. Notice that only one icon is used for all the ports supported by a port driver. This limitation is imposed to avoid confusing the user with too many different icons.

Parameters

Parameters	Data Types	Description
hab	HAB	Handle to anchor block
idIcon	PULONG	Resource ID of icon bit map

Return Codes: This handling routine returns FALSE if no icon is available. The system then uses a default port icon.

Remarks: The Workplace Shell calls SplQpQueryIcon to load and draw port icons, as appropriate.

SplPdInitPort

```
APIERR APIENTRY SplPdInitPort (hfile, pszPortName)
```

This function initializes a port on behalf of the spooler.

Parameters

Parameters	Data Types	Description
hfile	HFILE	Handle to an open file
pszPortName	PSZ	Name of port to be initialized

Return Codes: This handling routine returns the following errors:

ERROR_INVALID_PARAMETER
NO_ERROR.

Remarks: The port driver reads the initialization data from the INI file, interprets the data, and issues the appropriate DosDevIOCtls to the open port given by the file handle.

This function is called from the spooler function, PrtOpen, to complete port opening. Notice that PrtOpen issues the actual DosOpen first. PrtOpen holds a semaphore for each port in use and a linked list of threads waiting on the semaphore. This is because, although the spooler serializes output, it is still possible for applications to write directly to the port by using DevOpenDC (OD_DIRECT).

SplPdInstallPort

```
APIERR APIENTRY SplPdInstallPort (hab, pszPortName)
```

This function tells the port driver the name of the port that needs to be installed.

Parameters

Parameters	Data Types	Description
hab	HAB	Handle to anchor block
pszPortName	PSZ	Name of port to be installed

Return Codes: This handling routine returns the following errors:

ERROR_INVALID_PARAMETER
NO_ERROR.

Remarks: This function is called from the Workplace Shell. The port driver writes the initialization data from the INI file. The Workplace Shell then typically calls SplPdSetPort.

SplPdQueryPort

```
APIERR APIENTRY SplPdQueryPort (hab, pszPortName, pBuf, cbBuf, cItems)
```

This function returns textual data that describes the port configuration in a way that can be printed by the Workplace Shell.

Parameters

Parameters	Data Types	Description
hab	HAB	Handle to anchor block
pszPortName	PSZ	Name of port to be configured
pBuf	PVOID	Pointer to buffer of data structures
cbBuf	ULONG	Size of buffer in bytes
cItems	PULONG	Count of number of strings of description returned

Return Codes: This handling routine returns the following errors:

ERROR_INSUFFICIENT_BUFFER
NO_ERROR.

Remarks: The buffer consists of an array of strings (PSZ), the number of which is given by cItems. Each string contains one line of text that can be from 0 – 80 characters long. The port name and port description is not required; the Workplace Shell can retrieve these from the OS2SYS.INI file.

The maximum size of the data returned is limited to 4KB. If the buffer is too small, the error code ERROR_INSUFFICIENT_BUFFER is returned.

SplPdRemovePort

```
APIERR APIENTRY SplPdRemovePort (hab, pszPortName)
```

This function tells the port driver the name of the port that needs to be removed.

Parameters

Parameters	Data Types	Description
hab	HAB	Handle to anchor block
pszPortName	PSZ	Name of port to be removed

Return Codes: This handling routine returns the following errors:

ERROR_INVALID_PARAMETER
NO_ERROR.

Remarks: This function is called from the Workplace Shell and allows the port driver to remove its data from the INI file.

SplPdSetPort

```
APIERR APIENTRY SplPdSetPort (hab, pszPortName, flModified)
```

This function displays a dialog to allow the user to browse and modify port configurations.

Parameters

Parameters	Data Types	Description
hab	HAB	Handle to anchor block
pszPortName	PSZ	Name of port to be configured
flModified	PULONG	Flag to indicate that the configuration has been modified

Return Codes: This handling routine returns the following errors:

ERROR_INVALID_PARAMETER
NO_ERROR.

Remarks: This function is called from the Workplace Shell. The port driver retrieves the values previously stored in OS2SYS.INI and displays a dialog. Default values are used the first time if no previous values were stored in OS2SYS.INI.

When the user selects OK on the dialog box, the port driver stores the changed values back into OS2SYS.INI. The flag flModified is not set if the user did not modify anything, or if the user selected CANCEL on the dialog.

Note: The dialog should contain everything required for port initialization and termination.

SplPdTermPort

```
APIERR APIENTRY SplPdTermPort (hfile, pszPortName)
```

This function terminates a port on behalf of the spooler.

Parameters

Parameters	Data Types	Description
hfile	HFILE	Handle to an open file
pszPortName	PSZ	Name of the port to be terminated (closed)

Return Codes: This handling routine returns the following errors:

ERROR_INVALID_PARAMETER
NO_ERROR.

Remarks: The port driver reads the termination data from the INI file, interprets the data, and issues the appropriate DosDevIOCtls to the open port given by the file handle.

This function is called from the spooler function, PrtClose, to start port close down. Notice that PrtClose then issues a DosClose.

Part 4. Reference Material

Chapter 7. Exported Entry Points

This chapter describes the entry points that are exported by a presentation driver dynamic link library:

```
EXPORTS
   OS2_PM_DRV_RING_LEVELS      /* All drivers - Optional  */
   OS2_PM_DRV_ENABLE_LEVELS    /* All drivers - Optional  */
   OS2_PM_DRV_ENABLE           /* All drivers - Mandatory */
```

OS2_PM_DRV_RING_LEVELS

This entry point is the address of a table of ring levels required when dispatching each of the functions hooked in the dispatch table by "Enable Subfunction 01H — FillLogicalDeviceBlock" on page 7-6. The table is an array of bytes corresponding to the functions in the dispatch table, terminating with a byte of *0* (that is, an ASCIIZ string). The most significant six bits of each byte are reserved and must be *0*. The remaining two bits of each byte represent the ring level to be used when dispatching the corresponding function in the dispatch table.

```
00 = End of Table          10 = Ring 2
01 = Ring 2 Conforming     11 = Ring 3
```

Any function that does not have a corresponding byte in the table will be dispatched as Ring 2 Conforming. This is the most desirable case from the standpoint of system performance.

Function	Ring Level
GreGetArcParameters	0x01
GreSetArcParameters	0x01
GreArc	0x03
GrePartialArc	0x02
All others	0x00

The following table of 5 bytes would declare GreArc as Ring 3, GrePartialArc as Ring 2, and all other functions as Ring 2 Conforming:

```
0x01    0x01    0x03    0x02    0x00
```

If this table is not exported, all functions will be dispatched as Ring 2 Conforming as if the table had contained the single byte 0x00.

OS2_PM_DRV_ENABLE_LEVELS

This entry point is the address of a table of ring levels required when calling each of the Enable subfunctions. The table is an array of bytes corresponding to the subfunction numbers, terminating with a byte of *0* (that is, an ASCIIZ string). The most significant six bits of each byte are reserved and must be *0*. The remaining two bits of each byte represent the ring level to be used when dispatching the corresponding function in the dispatch table.

```
00 = End of Table          10 = Ring 2
01 = Ring 2 Conforming     11 = Ring 3
```

Any subfunction that does not have a corresponding byte in the table will be dispatched as Ring 2 Conforming. This is the most desirable case from the standpoint of system performance.

Subfunction	Ring Level
Unused	0x01
FillLogicalDeviceBlock	0x01
FillPhysicalDeviceBlock	0x03
Unused	0x01
DisablePhysicalDeviceBlock	0x02
All others	0x00

The following table of 6 bytes would declare FillPhysicalDeviceBlock as Ring 3, DisablePhysicalDeviceBlock as Ring 2, and all other subfunctions as Ring 2 Conforming:

```
0x01    0x01    0x03    0x01    0x02    0x00
```

If this table is not exported, all subfunctions will be called as Ring 2 Conforming as if the table had contained the single byte 0x00.

OS2_PM_DRV_ENABLE

The enable entry point is exported as OS2_PM_DRV_ENABLE by the presentation driver. The Enable routine in the driver supports a set of subfunctions that enable or disable the environment for a device context owned by a specific application or process. When a device context is opened or closed, the Presentation Manager interface issues a series of calls to subfunctions at the enable entry point. These calls initialize the presentation driver, the physical device, and the device context.

The syntax used by the Presentation Manager interface to call the Enable routine is:

```
LONG APIENTRY OS2_PM_DRV_ENABLE (Subfunc, Param1, Param2)

ULONG   Subfunc;
ULONG   Param1;
ULONG   Param2;
```

Note: LONG, ULONG (unsigned LONG), and APIENTRY are defined in OS2DEF.H, which is included through the header file OS2.H.

Stack Frame: At entry to the Enable routine, the stack frame contains:

Parameter	Description
Subfunction	32-bit value identifying the subfunction
Param1	First parameter
Param2	Second parameter

Subfunctions: Presentation drivers must support the following Enable subfunctions:

01H FillLogicalDeviceBlock
02H FillPhysicalDeviceBlock
04H DisablePhysicalDeviceBlock
05H EnableDeviceContext
06H DisableDeviceContext
07H SaveDCState
08H RestoreDCState
09H ResetDCState
0AH CompleteOpenDC
0BH BeginCloseDC.

Note: The Enable function should return –1 (ERROR_MINUS) for Subfunction 03H, and numbers greater than 0BH.

Device Context Management: Device contexts are opened in response to an application or process calling the DevOpenDC API function. On receiving this call, Presentation Manager loads the presentation driver (if it is not already present and able to support the device context) and issues a series of calls to the enable entry point.

Initially, when the presentation driver has not been enabled for use by the calling application or process, the driver receives the series of calls shown in Figure 7-1. When the driver has been enabled for an application or process, the sequence of calls to enable additional device contexts does not include FillLogicalDeviceBlock and, depending upon a requirement that the driver posts in the initial enable sequence, includes or excludes FillPhysicalDeviceBlock.

The OS2_PM_DRV_ENABLE entry point handles the DC management functions. Those functions can be placed in three categories:

- Transactions involved in opening a DC
- Transactions involved in closing a DC
- Other DC functions including SaveDC, RestoreDC and ResetDC.

Opening and Closing DCs: The open and close transaction sequences are symmetrical:

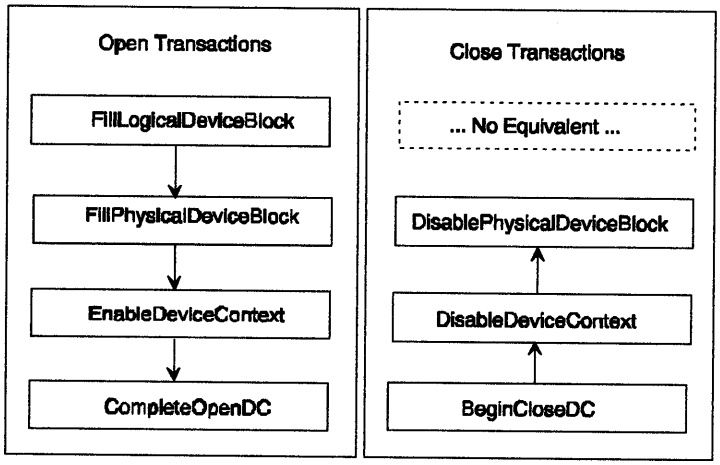


Figure 7-1. Enabling a Presentation Driver

Device contexts are opened in response to an application or process calling the DevOpenDC API function. On receiving this call, the graphics engine (PMGRE.DLL) loads the presentation driver, if it is not already present and able to support the device context, and issues a series of calls to the OS2_PM_DRV_ENABLE entry point.

Initially, when the presentation driver has not been enabled for use by the calling application or process, the driver receives the Open Transaction series of calls shown in Figure 7-1. When the driver has been enabled for an application or process, the sequence of calls to enable additional device contexts can include FillLogicalDeviceBlock and, depending upon a requirement that the driver posts in the initial enable sequence, includes or excludes FillPhysicalDeviceBlock.

Each of the Close Transactions functions undoes actions taken by a corresponding Open Transaction function. For example, the EnableDeviceContext function allocates the DDC data area within the presentation driver and returns the hDDC handle. The DisableDeviceContext function deallocates the DDC data area.

The FillLogicalDeviceBlock transaction occurs when a thread of a process opens the first instance of a DC for this presentation driver. During this transaction, a dispatch table will be created for later use by the graphics engine.

The FillPhysicalDeviceBlock transaction will occur only if the result flags from the FillLogicalDeviceBlock transaction indicate that it is necessary. See the description of "Bit 2" in the "Enable Subfunction 02H – FillPhysicalDeviceBlock" on page 7-8.)

This subfunction is used by presentation drivers which support multiple types of physical devices. It will occur during the DevOpenDC call for every DC associated with a particular type of physical device. (This is where any memory allocation and initializations relating to a physical device should be done.) During the EnableDeviceContext transaction, the presentation driver will allocate and format the driver-specific data (DDC) associated with the DC being constructed. At CompleteOpenDC time, the DC is completely constructed and initialized. This transaction gives the presentation driver an opportunity to make final adjustments to the DC before the application gets access to it.

If either the FillLogicalDeviceBlock or FillPhysicalDeviceBlock transactions return a failure return code, no other transactions are requested by the graphics engine.

If the EnableDeviceContext fails, the DisablePhysicalDeviceBlock transaction is sent to the presentation driver. If the CompleteOpenDC transaction fails, the BeginCloseDC, DisableDeviceContext, and DisablePhysicalDeviceBlock transactions are sent to the presentation driver.

If an application has opened a device context but the application terminates before it closes the device context, the graphics engine still calls the disable routines BeginCloseDC, DisableDeviceContext, and DisablePhysicalDeviceBlock as expected by the presentation driver when a device context is closed explicitly by an application.

Saving and Restoring DCs: The SaveDCState and RestoreDCState functions manage a stack of entries that correspond to most of the state of a DC. SaveDCState pushes an entry on the stack. RestoreDCState retrieves one of the pushed entries and uses it to revert the DC back to its earlier saved state. Notice that a particular RestoreDCState call can pop more than one entry from the stack of SaveDC entries.

Reset DC: This call resets a DC to its original initialized state as it was just after the CompleteOpenDC transaction.

Enable Subfunction 01H – FillLogicalDeviceBlock

This subfunction is called on the first occasion that a specific application or process opens a device context that uses the called presentation driver. It is not called when the same application or process opens additional DCs with the same presentation driver. However, the graphics engine can call this subfunction at other times and the call should always be honored. If the value of pDispatchTable is non-zero, the handling routine must initialize the dispatch table.

The major tasks that the handling routine must implement are:

1. Add an entry to the DosExitList to ensure that any allocated resources are freed when the owning application or process terminates.
2. Save those pointers in the dispatch table that are needed to pass hooked functions back to the graphics engine. A typical example is GreCharString, which must be hooked by the presentation driver but can be passed back to the default handling routine.
3. Initialize the dispatch table. That is, modify the table so that the entries for functions hooked by the presentation driver contain pointers to the driver's handling routines.
4. Set flags to indicate how future DevOpenDC calls to this device should be handled.

In some typical presentation drivers, the handling routine for FillLogicalDeviceBlock allocates global heap space for use by the device contexts. The memory for this heap space is obtained by calling the SSAllocMem function described in Chapter 12, System Functions. This global heap space is available to all instances of a DC that are opened by the application or process for which the presentation driver was enabled.

Stack Frame

Parameter	Description
ulSubfunction	01H.
pParam1	Pointer to a structure. See below.
pParam2	Pointer. See below.
pDispatchTable	Pointer to the presentation driver's copy of the dispatch table. This table is an array of 32-bit pointers to system-default function handling routines. The low-order byte of a function number identifies the offset to the relevant pointer.

pParam1 Pointer to the following structure:

ulVersion Version number in binary coded decimal (BCD) of the graphic engine.

cTableSize The number of entries in the dispatch table. The presentation driver should not replace pointers past the end of the table.

pParam2 Pointer to the following structure:

pfsFlags Pointer to a WORD of logical device flags. Flag bits, 0 and 2, apply to the presentation driver and is set *on* or *off*, as appropriate, by the presentation driver. All other flags are reserved for system use and must not be modified.

Bit 0 Set *on*, if each DC for this presentation driver requires its own physical device block. Even if the presentation driver sets this bit *off*, the FillPhysicalDeviceBlock subfunction can still be called more than once (though no work might need to be done after FillPhysicalDeviceBlock is called the first time).

Bit 2 Set *on*, if the szDeviceName and pszDriverName fields of a DevOpenDC call for this device are ignored. Setting bit 2 *on* indicates that the presentation driver supports only one physical device in one configuration. For example, the display driver. Hardcopy drivers do *not* set this bit unless the hardcopy driver supports only one physical device in one configuration.

Return Codes: The handling routine should return a LONG integer. Valid values are:

0 Successful
–1 Error

Enable Subfunction 02H – FillPhysicalDeviceBlock

This subfunction is always called in the initial set of calls to the enable function for a specific application or process. It is called only when additional device contexts are enabled if the value of flag bit 0, as set in the initial call to FillLogicalDeviceBlock, shows that each device context requires its own physical device block. However, the graphics engine can call this subfunction at other times. If it is called more than once when the individual DCs do not require separate physical device blocks, the handling routine does nothing and returns the existing ulStateInfo handle or pointer from the DC instance data.

Print jobs must be spooled using the data type given on the call to Enable subfunction 02H –FillPhysicalDeviceBlock). Therefore, print jobs queued PM_Q_STD can be printed with queue processor options applied to the job.

The physical device block is located in the driver's global heap. To initialize the block, the presentation driver uses:

- Default values set by the presentation driver
- Values read from the initialization file
- Values from the DEVOPENSTRUC structure.

A typical physical device block is shown under "Remarks" below.

Stack Frame

Parameter	Description
ulSubfunction	02H.
pParam1	Pointer to an extended DEVOPENSTRUC structure. See below.
ulParam2	Reserved.

pParam1 Pointer to an extended DEVOPENSTRUC structure (the structure is extended by adding a DENPARAMS structure to give three additional fields, ulStateInfo, ulType, and ulHDC):

pszLogAddress Pointer to the logical address of the device. For example, 'LPT1Q'.

pszDriverName Pointer to name of the presentation driver. For example, 'LaserJet**'.

pdriv Pointer to DRIVDATA structure. This structure contains data generated by the presentation driver during the dialog that set device modes. See "OS2_PM_DRV_DEVMODE" on page 4-2.

cb	Number of bytes in the structure
lVersion	Version number
szDeviceName[32]	Device name
abGeneralData	Driver-specific data.

pszDataType Pointer to the name of the queue file data type:

PM_Q_STD
PM_Q_RAW

pszComment Pointer to a file description that the spooler could use in messages displayed to the user; usually the name of the application by convention.

** Trademark of the Hewlett-Packard Company.

pszQueueProcName Pointer to name of queue driver (queue processor).

pszQueueProcParams Pointer to a string of queue processor parameters.

pszSpoolerParams Pointer to a string of spooler parameters separated by one or more blanks.

FORM = aaa Identifies the form name for the print job. Multiple names are separated by commas (aaa,bbb,ccc). If this parameter is not present, the job is printed on the current form.

Form names are defined by the presentation driver. Valid names are those that are returned from a call to the driver's GreQueryHardcopyCaps handling routine.

pszNetworkParams Pointer to a string of networking parameters.

ulStateInfo Reserved. This field does not contain meaningful information at this time.

ulType DC type (for example, OD_QUEUED).

ulHDC DC handle.

Return Codes: The handling routine should return a LONG integer. Valid values are:

–1 Error.

Other Handle or pointer, ulStateInfo, to the physical device block. This pointer is passed back to the presentation driver in subsequent calls to the Enable subfunctions, 04H and 05H.

Note: The name, pDCI, is used in place of ulStateInfo in some source code. See "Remarks" below.

Remarks: Physical device blocks hold information about the presentation driver and the device that is the same for every instance of a device context. The design of the presentation driver determines what information is held in the physical device block. A typical printer physical device block is given below:

```
typedef struct
{
    Dptr                PDBDriverName;              /* String for driver name       */
    Dptr                PDBOutputName;              /* String for output name       */
    unsigned            PDBHandle;                  /* Handle for DOS device        */
    WORD                PDBOutputType;              /* Type of output; STD/ESC/RAW  */
    DeviceSemaphoreTable PDBDevice;                 /* Pointer to device table      */
    Byte                PDBScratch[DCT_MAX_SIMPLE]; /* Scratch pad for printer      */
    DDTType             DDT;                        /* Copy of the DDT to be used   */
    WORD                RasterMode;                 /* Raster type used             */
    WORD                PrintQuality;               /* Draft or LQ printing         */
    WORD                Orientation;                /* Portrait or landscape        */
    DWORD               Version;                    /* Driver version number        */
    DevRect             FormClipRegion;             /* Clip region for current form */
    Byte                DeviceName[32];             /* Model number, and so forth.  */
} PDBIType;
```

Hardcopy drivers must be able to accept a UNC queue name as a logical address on Enable subfunction 02H – FillPhysicalDeviceBlock. The UNC queue name is passed to the SplQmOpen API which handles rerouting the spool job across the LAN.

Some applications use the SpoolerParams to submit form names. However, the forms can also be supplied in driverdata. SpoolerParams take precedence over the driver data. At Enable subfunction 02H – FillPhysicalDeviceBlock time, any SpoolerParams are integrated with the *pdriv* field of the DevOpenStruc.

If *pdriv* is NULL, it is created by obtaining the data from the system. The hardcopy driver attempts to find a printer in the INI file that uses this port name, and derives job properties from the device defaults and printer properties.

Note: Hardcopy drivers process all fields except pszQueueProcName, pszQueueProcParams, and pszNetworkParams.

Enable Subfunction 04H – DisablePhysicalDeviceBlock

This subfunction in the presentation driver is called by the system to disable the specified device and to free any associated memory. Presentation drivers for the primary display device return a value of *0* without taking any action.

Note: The operating system never calls this subfunction in the hardcopy driver if the driver uses one physical device block to support multiple device contexts. Presentation drivers notify the operating system of this capability by *not* setting bit 0 of the Logical Device flags returned to the system from the FillLogicalDeviceBlock subroutine.

Stack Frame

Parameter	Description
ulSubfunction	04H.
ulParam1	Handle or pointer, ulStateInfo, to physical device block.
pParam2	Not used.

Return Codes: The handling routine should return a LONG integer. Valid values are:

0 Successful
–1 Error

Enable Subfunction 05H – EnableDeviceContext

In response to this subfunction, the driver reserves the memory it needs to support this instance of a DC and initializes the instance data. The value of the return code from this subfunction, if it is not *–1*, is saved by the system and passed back to the presentation driver on all future calls to this DC instance. This value is expected to be a pointer or handle to the instance data. Instance data is described under "Remarks" on page 7-13.

The handling routine initializes relevant fields in the instance data to their default values. For example, it calls WinQueryProcessCP to get the initial Code Page ID.

STARTDOC/ABORTDOC/ENDDOC (Hardcopy drivers only): The hardcopy driver allows the following sequences to GreEscapes within Enable subfunction 05H – EnableDeviceContext and Enable subfunction 0BH – BeginCloseDC brackets:

GreEscape DEVESC_STARTDOC
GreEscape DEVESC_ENDDOC – spools first job
GreEscape DEVESC_STARTDOC
GreEscape DEVESC_ENDDOC – spools second job
GreEscape DEVESC_STARTDOC
GreEscape DEVESC_ENDDOC – etc.

GreEscape DEVESC_STARTDOC
GreEscape DEVESC_ABORTDOC – aborts job
GreEscape DEVESC_STARTDOC
GreEscape DEVESC_ENDDOC – spools first job.

Stack Frame

Parameter	Description
ulSubfunction	05H.
pParam1	Pointer to a DENPARAMS structure. See below.
pParam2	Not used.

pParam1 Pointer to a DENPARAMS structure:

ulStateInfo Value returned by FillPhysicalDeviceBlock.

ulType Type of device context. Defined values are:

OD_QUEUED
OD_DIRECT
OD_INFO
OD_MEMORY.

For details, see DevOpenDC in the *OS/2 2.0 Presentation Manager Programming Reference*.

ulHDC Device context handle.

Return Codes: The handling routine should return a LONG integer. Valid values are:

–1 Error.
other Pointer (pInstance) to the DC instance data.

Remarks: Instance data refers to information (such as the name of a spool file and whether a bit map has been created) that applies to a specific instance of a device context. Global data, which applies to all instances of a device context that is using the same presentation driver, is held in the Physical Device Block (PDB). A pointer to this block is passed in the DENPARMS structure to the EnableDeviceContext routine to be included in the instance data. See "Instance Data" on page 1-9.

A typical example of instance data follows:

```
typedef struct _DC { /* dc */
    USHORT          DCIIdentifier;     /* Contains DC_IDENTIFIER              */
    ULONG           DCIDCType;         /* DC type 0-8                         *
    HDC             DCIhdc;            /* DC handle                           */

    struct _DC *    DCINextEntry;      /* Next instance                       */
    LONG            DCISaveCount;      /* Number of saved DC states           */

    pBitmapHeader   DCISelListEntry;   /* Selected bit-map list entry         */
    USHORT          DCIBitmapType;     /* Current bit-map type                */

    POINTL          DCICurrPos;        /* Current position                    */

    DCHARBUNDLE     DCICurTxtAts;      /* Current Text attributes bundle      */
    DLINEBUNDLE     DCICurLinAts;      /* Current Line attributes bundle      */
    DAREABUNDLE     DCICurPtnAts;      /* Current Pattern attributes bundle   */
    DIMAGEBUNDLE    DCICurImgAts;      /* Current Image attributes bundle     */
    DMARKERBUNDLE   DCICurMrkAts;      /* Current Marker attributes bundle    */

    USHORT          DCILinePatCnt;     /* Current line pattern count          */
    USHORT          DCILineTypMask;    /* Mask used for line types            */

    POINTL          DCIPatternOrigin;  /* Pattern origin                      */

    pBitmapHeader   DCIMarker;         /* Pointer to marker definition        */
    BOOL            DCIMarkerSimReq;   /* On, if simulation required          */

    USHORT          DCIFontType;       /* Type of current font                */

    pRealizedFontType DCIRealizedFonts; /* Font table array                   */
    USHORT          DCIFontTabNum;     /* Number of Font table entries        */
    BOOL            DCIPairKerning;    /* Enabled\disabled state              */

    ULONG           DCICodePage;       /* Currently selected code page        */

    BOOL            DCITextSim;        /* Set if text simulation required     */

    POINTL          DCIOrigin;         /* Current DC origin                   */

    PCOLORTABLE     DCIColorTable;     /* Pointer to Color table              */
    BOOL            DCIBackgndDefined; /* Status of CLR_BACKGROUND            */
    BOOL            DCINeutralDefined; /* Status of CLR_NEUTRAL               */
    USHORT          DCIColTabSize;     /* Color table size                    */
    USHORT          DCIColFormat;      /* Format of color table               */
    USHORT          DCIColStatus;      /* Status of color table               */
    USHORT          DCILowIndex;       /* Lowest index in table               */
    USHORT          DCIHighIndex;      /* Highest index in table              */
    USHORT          DCISysColState;    /* Latest state of system colors       */

    COLORPAIR       DCILineColatts;    /* Line color indexes                  */
    COLORPAIR       DCIPattColatts;    /* Pattern colors indexes              */
    COLORPAIR       DCICharColatts;    /* Character colors indexes            */
    COLORPAIR       DCIImagColatts;    /* Image colors indexes                */
    COLORPAIR       DCIMarkColatts;    /* Marker colors indexes               */

    ClipRectangle   DCIClipRects[CACHED_CLIPS];
                                       /* Clip rectangles                     */
    BOOL            ClipChanged;       /*                                     */
    USHORT          DCIClipOrder;      /* Order of clip rectangles            */
    USHORT          DCIClipNum;        /* Number of clip rectangles           */
    RECTL           DCIBoundingClip;   /* Current screen/bit-map area         */
    USHORT          DCIEngineClips;    /* Clip regions in engine              */
    BOOL            DCIIsDirty;        /* Indicates when clip regions are invalid */

    RECTL           DCIGPIBounds;      /* Current GPI bounds                  */
```

```
    BOOL            DCIDefGPIBounds;   /* On, if GPI bounds are default          */
    RECTL           DCIUserBounds;     /* Current user bounds                    */
    BOOL            DCIDefUserBounds;  /* On, if user bounds are default         */
    SHORT           DCIConvFactor;     /* For device, conversion                 */

    BOOL            DCICorrInvalid;    /* Correlation rectangles invalid         */
    RECTL           DCIPickWindowPage; /* Pick window in page coordinates        */
    DevRect         DCIPickWindowDevice;
                                       /* Pick window in device coordinates      */
    PRECTL          DCICorrRects;      /* Current correlation rectangles         */
    USHORT          DCICorrNum;        /* Number of correlation rectangles       */
    USHORT          DCICorrSize;       /* Number of correlation rectangles for   */
                                       /* which storage has been allocated       */


    XFORM           DCITransform;      /* Transformation data                    */

    FONTDETAILS     DCIAvioFonts[CNT_LOADABLE_LCIDS + 1];
                                       /* AVIO loadable font definitions         */

    USHORT          DCIChanged;        /* Shows non-default bundles              */

    USHORT          DCISpacingType;    /* Spacing type of current font           */
    BOOL            DCIXFrmSimple;     /* Transform type                         */
    ULONG           StyleNumber;       /* Line style state and mask              */

    USHORT          DCICommandMask;    /* Used to mask command bits              */

    POINTL          DCICurrPosWorld;   /* Current world position                 */
    pBitmapHeader   Pattern;           /* Pattern for direct DC                  */

    AVIOINFO        DCIAvioInfo;       /* Avioparms material                     */
    FONTDETAILS     CurrentFont;       /* Presentation Manager format current font */
    PDEVPAL         Palette;           /* Handle/pointer of palette              */
    PRGB2           DCIDeviceDefaultPalette;
                                       /* Device default palette that was current */
                                       /* the last time color table indexes were */
                                       /* calculated                             */
} DC;
```

Enable Subfunction 06H – DisableDeviceContext

This subfunction is called when a device context is about to be deleted. In response, the presentation driver must release any memory and other resources that it has allocated for the DC. The presentation driver uses the DC instance data to identify this memory.

Stack Frame

Parameter	Description
ulSubfunction	06H
pParam1	Pointer (pInstance) to the DC instance data
pParam2	Not used

Return Codes: The handling routine should return a LONG integer. Valid values are:

0 Successful
–1 Error

Enable Subfunction 07H – SaveDCState

This subfunction requests the presentation driver to save all of the information that it has about the device context. The state of a DC might be saved multiple times in *last in, first out (LIFO)* order. The routine returns an error code if there is not enough memory available to save the state.

Note: The handling routine must keep a count of the number of saved states.

Stack Frame

Parameter	Description
ulSubfunction	07H
pParam1	Pointer (pInstance) to the DC instance data
pParam2	Not used

Return Codes: The handling routine should return a LONG integer. Valid values are:

0 Successful
–1 Error

Enable Subfunction 08H – RestoreDCState

This subfunction restores a specific DC state from the saved DC states. An index to the required state is supplied as Parameter 2. The presentation driver returns an error if the index is zero, or if it specifies a value that does not identify a saved state.

Display drivers must be careful when handling this call. Some of the data stored in the DC instance data must match the data held by the Window Manager. The handling routine for RestoreDCState does not restore:

- DC origin
- User bounds
- Cached clipping rectangles
- HDC_IS_DIRTY flag.

Stack Frame

Parameter	Description
ulSubfunction	08H.
pParam1	Pointer (pInstance) to the DC instance data.
lParam2	Identifies which of the saved states are to be restored. See below.

lParam2 Identifies which of the saved states are to be restored. Positive numbers indicate the specific state counting from the first saved state, that is, 1 equals the first, 2 the second, and so forth. All saved states following the one being restored are discarded. Preceding states remain saved.

Negative numbers indicate the specific state counting from the last saved state, a value of –2 indicates that the last state is discarded and the state before that is restored.

Return Codes: The handling routine should return a LONG integer. Valid values are:

0 Successful
–1 Error

Enable Subfunction 09H – ResetDCState

This subfunction resets the device context to its original initialized state. The presentation driver deletes all fonts, patterns, and paths, and resets all attributes to their default values. Notice that when resources are not owned by the presentation driver, the driver saves the relevant values so that they are available to reset the device context. A typical example is when the default font is a graphics engine font. In this case, the presentation driver saves the font flags and address passed in the first call to GreDeviceSetAttributes.

The *visible region* and the DC origin are not affected by this function.

Stack Frame

Parameter	Description
ulSubfunction	09H.
pParam1	Pointer (pInstance) to the DC instance data.
ulParam2	Reserved.

Return Codes: The handling routine should return a LONG integer. Valid values are:

0 Successful
–1 Error

Enable Subfunction 0AH – CompleteOpenDC

This subfunction is called upon completion of the DevOpenDC process to tell the presentation driver that the device context now has access to the graphics engine. Presentation drivers for the primary display device return Successful without taking any action. For other devices, the handling routine in the presentation driver completes the initialization process for any resources, such as bit maps, that are obtained through calls to the graphics engine.

Hardcopy drivers do not use the CompleteOpenDC routine to open resources such as spool files or journal files. If these resources are required, they are opened in response to a call to GreEscape with DEVESC_STARTDOC (see page 8-81). Such drivers set a flag in the instance data to show that DEVESC_STARTDOC has been received, and do not process any output until that flag has been set.

Note: The presentation driver locks and unlocks resources such as bit maps and device contexts to prevent simultaneous use of the resource by two threads belonging to the same process. Typically, this is done by setting a semaphore or some other form of *busy* flag for the resource.

Stack Frame

Parameter	Description
ulSubfunction	0AH
hdcParam1	Device context handle
pParam2	Pointer (pInstance) to the DC instance data

Return Codes: The handling routine should return a LONG integer. Valid values are:

0 Successful
–1 Error

Enable Subfunction 0BH – BeginCloseDC

This subfunction is called to inform the presentation driver that the device context is being closed. This is the last call made to the driver before it loses access to the graphics engine. Display drivers for the primary display device, return Successful without taking any action. For other devices, the handling routine in the presentation driver has to close any resources (such as journal files and bit maps) that it owns.

Hardcopy drivers do not use the BeginCloseDC routine to complete tasks such as writing spool files. The tasks are completed in response to a call to GreEscape with DEVESC_ENDDOC (see page 8-70). The DEVESC_ENDDOC routine resets the DEVESC_STARTDOC flag in the instance data. The BeginCloseDC routine checks that the flag is reset before taking any action.

Stack Frame

Parameter	Description
ulSubfunction	0BH
hdcParam1	Device context handle
pParam2	Pointer (pInstance) to the DC instance data

Return Codes: The handling routine should return a LONG integer. Valid values are:

0 Successful
–1 Error

Chapter 8. Mandatory Functions for All Drivers

This chapter describes those functions, which must be supported for all devices, that are called through the dispatch table by handling routines in the presentation driver. The dispatch table contains the address of each function that the presentation driver can hook. Initially, when the presentation driver is first loaded, a copy of the default dispatch table is passed to the driver. The presentation driver's Enable routine modifies this copy so that the entries for functions supported in the driver point to the handling routines in the driver. Entries to the table that are to be modified can first be saved by the presentation driver in case they are subsequently needed. The original saved table entries, and those that are not modified, point to the engine-simulation routines for the functions concerned.

Functions listed in the dispatch table fall into two categories :

- Functions that the presentation driver *must* support (mandatory functions)
- Functions that are supported by the graphics engine but can be optionally hooked by the presentation driver.

Descriptions are provided of the mandatory functions. Each description shows what the handling routine is expected to do, the parameters passed to the routine, and the values that the routine returns. The functions are grouped according to the conditional include sections of the header file:

- Attribute functions (INCL_GRE_DEVMISC1)
- Bit-map functions (INCL_GRE_BITMAPS)
- Color table functions (INCL_GRE_COLORTABLE)
- Device functions 2 (INCL_GRE_DEVMISC2)
- Device functions 3 (INCL_GRE_DEVMISC3)
- GreEscape functions (INCL_GRE_DEVICE)
- Line functions (INCL_GRE_LINES, INCL_GRE_SCANS)
- Marker functions (INCL_GRE_MARKERS)
- Query functions (INCL_GRE_DEVICE)
- Text (String) functions (INCL_GRE_STRINGS).

Additional functions that must be supported by presentation drivers for display devices are described in Chapter 9, "Mandatory Functions for Display Drivers." Hardcopy drivers must also provide some support for these display device functions. This can be a common routine that returns Successful and posts the warning, PMERR_DEV_FUNC_NOT_INSTALLED.

The level of support provided in the handling routine for a particular function depends on the type of device and the level of support provided for that device. At the minimum, the handling routine must indicate a successful completion.

Attribute and Bundle Definitions

A general description of colors, mixes, patterns, and the attribute definitions for each attribute bundle type are provided. For area definitions, the area must be filled by using the pattern that is current when GreBeginArea is called.

Colors

All colors are passed as 32-bit signed values. These are either indexes into the Logical Color table, logical palette indexes, or representations of 24-bit Red, Green, and Blue (RGB) values. Some special attribute values can be passed to the graphics engine and returned by GreGetAttributes:

CLR_FALSE All color planes or bits, or both, are *0*.

CLR_TRUE All color planes or bits, or both, are *1*.

CLR_WHITE This index is never loaded explicitly. It always produces White when the default color table is in force or when the index is set to RGB. With a realized color table and an index that is not RGB, the value CLR_WHITE produces the background color CLR_BACKGROUND.

CLR_BLACK This index is never loaded explicitly. It always produces Black when the default color table is in force or when the index is set to RGB. With a realized color table and an index that is not RGB, CLR_BLACK produces the neutral color CLR_NEUTRAL. See "Color Functions" on page 8-13.

CLR_FALSE and CLR_TRUE provide useful operands for Bitblt logical operations. CLR_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

Mix Modes

All values are passed to the graphics engine which passes them unchanged to the presentation driver.

Foreground Mix Mode Valid values are:

FM_OR	OR
FM_OVERPAINT	Overpaint
FM_XOR	Exclusive-OR
FM_LEAVEALONE	Leave alone (invisible)
FM_AND	AND
FM_SUBTRACT	(Inverse source) AND destination
FM_MASKSRCNOT	Source AND (inverse destination)
FM_ZERO	All zeros
FM_NOTMERGESRC	Inverse (source OR destination)
FM_NOTXORSRC	Inverse (source exclusive-OR destination)
FM_INVERT	Inverse of destination
FM_MERGESRCNOT	Source OR (inverse destination)
FM_NOTCOPYSRC	Inverse of source
FM_MERGENOTSRC	(Inverse source) OR destination
FM_NOTMASKSRC	Inverse of (source AND destination)
FM_ONE	All 1s.

Note: FM_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

The presentation driver must support FM_OR, FM_OVERPAINT, FM_XOR, and FM_LEAVEALONE. Other foreground mixes can be handled as FM_OVERPAINT. Mixing, other than for FM_LEAVEALONE or FM_OVERPAINT, is performed on the physical color index or logical palette index. When an indexed color table or palette has been realized, this corresponds to the logical color index. In other cases, the color resulting from a mix cannot be predicted.

Note: An exception to this rule occurs when the same object is drawn twice with FM_XOR and BM_LEAVEALONE, and no intermediate drawing is performed in other mixes. The implementation must guarantee that this always results in the object being erased cleanly.

Other valid mixes can also be supported for some primitive types. When a valid mix is not supported, the default is FM_OVERPAINT. An error is raised only when the specified mix value is not one of those listed above.

Background Mix Mode Valid values are:

BM_ERROR	Error
BM_DEFAULT	Default

BM_OR OR
BM_OVERPAINT Overpaint
BM_XOR. Exclusive-OR
BM_LEAVEALONE Leave alone (invisible).

The presentation driver must support BM_OVERPAINT and BM_LEAVEALONE. Other background mixes can be handled as BM_LEAVEALONE.

GreQueryDeviceCaps (see page 8-111) allows the application to determine the mixes supported by the device.

Line Attributes

The device line attributes are bundled in a DLINEBUNDLE structure:

Parameter	Description
cAttr	Size of the logical attribute bundle.
cDefs	Size of the LINEDEFS structure.
lbnd	LINEBUNDLE structure. The logical line bundle as seen by the application. See below.
ldef	LINEDEFS structure: **defType** Line definition. This is ignored by the presentation driver.

LINEBUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the LINEBUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
LBB_COLOR	lColor
LBB_BACK_COLOR	lBackColor
LBB_MIX_MODE	usMixMode
LBB_BACK_MIX_MODE	usBackMixMode
LBB_WIDTH	fxWidth
LBB_GEOM_WIDTH	lGeomWidth
LBB_TYPE	usType
LBB_END	usEnd
LBB_JOIN	usJoin.

lbnd The fields of a LINEBUNDLE structure are:

lColor Line foreground color.

lBackColor Line background color.

usMixMode Line foreground mix mode.

usBackMixMode Line background mix mode.

fxWidth Line-width multiplier. This value is expressed in fixed-point notation with a notational binary point between the second and third bytes. Therefore, 1.0 is represented by 10000H (or 65536). This multiplier is applied to the normal line width. Valid values are:

LINEWIDTH_DEFAULT	Default line width. This is the same as LINEWIDTH_NORMAL.
LINEWIDTH_NORMAL	Normal line width.
LINEWIDTH_THICK	Thick line width. This is double the normal width.

Typically, values equal or less than *1.0* are treated as LINEWIDTH_NORMAL and values greater than *1.0* as LINEWIDTH_THICK.

lGeomWidth Geometric line thickness in world-coordinate space specified as an integer value. This is used only by GreStrokePath or when MPATH_STROKE is specified for GreModifyPath. A value of *0* results in the thinnest line possible regardless of the transform in force. Thick geometric lines are treated as polygons and are transformed accordingly.

usType Specifies the cosmetic line type. Valid values are:

LINETYPE_DOT	Dotted
LINETYPE_SHORTDASH	Short-dashed
LINETYPE_DASHDOT	Dash, dot
LINETYPE_DOUBLEDOT	Double-dotted
LINETYPE_LONGDASH	Long-dashed
LINETYPE_DASHDOUBLEDOT	Dash, double-dot
LINETYPE_SOLID	Solid
LINETYPE_INVISIBLE	Invisible
LINETYPE_ALTERNATE	Every alternate pel on.

Note: LINETYPE_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

usEnd Valid values are:

LINEEND_FLAT	Flat
LINEEND_SQUARE	Square
LINEEND_ROUND	Round.

Note: LINEEND_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

usJoin Valid values are:

LINEJOIN_BEVEL	Bevel
LINEJOIN_ROUND	Round
LINEJOIN_MITRE	Miter.

Note: LINEJOIN_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

When lines join at a very acute angle and a mitered joint has been specified, then the length of the miter line could extend almost to infinity. To prevent this, when the ratio of miter length to geometric line width exceeds 10:1, a bevel joint is drawn. The miter length is the distance between the inner and outer intersection points.

When a wide line is explicitly closed by a call to GreCloseFigure from within a path, the style at the closure point is JOIN style not END style. If enough points are given to implicitly close the figure, the END style is used at the closure points. Notice that the LINEJOIN attribute is only to be used at wide line ends when the figure has been closed by a call to GreCloseFigure.

Area (Pattern) Attributes

The device area (pattern) attributes are bundled in a DAREABUNDLE structure:

Parameter	Description
cAttr	Size of the logical area bundle.
cDefs	Size of the AREADEFS structure. This is *0* when no device area bundle exists.
abnd	AREABUNDLE structure. See below.
adef	AREADEFS structure. See below.

AREABUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the AREABUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
ABB_COLOR	lColor
ABB_BACK_COLOR	lBackColor
ABB_MIX_MODE	usMixMode
ABB_BACK_MIX_MODE	usBackMixMode
ABB_SET	usSet
ABB_SYMBOL	usSymbol
ABB_REF_POINT	ptlRefPoint

abnd The fields of an AREABUNDLE structure are:

lColor Area foreground color.

lBackColor Area background color.

usMixMode Area foreground mix mode.

usBackMixMode Area background mix mode.

usSet Local identifier (lcid) for a logical font or a bit map. Valid values are:

0	Base pattern set
Non-zero	Local identifier for the logical font or bit map defined by the *cdef.defSet* field in the area attributes bundle.

usSymbol Identity of the required pattern in the current pattern set or logical font. This attribute is ignored when the pattern set is a bit map. If the value is outside the range of the logical font, the standard default pattern is used.

Values in the range 1 – 255 are valid. The defined values are:

PATSYM_ERROR	Error
PATSYM_DEFAULT	Default
PATSYM_DENSE1	Solid shading with decreasing intensity
PATSYM_DENSE2	Solid shading with decreasing intensity
PATSYM_DENSE3	Solid shading with decreasing intensity
PATSYM_DENSE4	Solid shading with decreasing intensity
PATSYM_DENSE5	Solid shading with decreasing intensity
PATSYM_DENSE6	Solid shading with decreasing intensity
PATSYM_DENSE7	Solid shading with decreasing intensity

PATSYM_DENSE8	Solid shading with decreasing intensity
PATSYM_VERT	Vertical lines
PATSYM_HORIZ	Horizontal lines
PATSYM_DIAG1	Diagonal lines 1, bottom-left to top-right
PATSYM_DIAG2	Diagonal lines 2, bottom-left to top-right
PATSYM_DIAG3	Diagonal lines 1, top-left to bottom-right
PATSYM_DIAG4	Diagonal lines 2, top-left to bottom-right
PATSYM_NOSHADE	No shading
PATSYM_SOLID	Solid shading
PATSYM_BLANK	Blank (same as PATSYM_NOSHADE)
PATSYM_HALFTONE	Every alternate pel *on*. PATSYM_HALFTONE can be similar to PATSYM_DENSE4 and PATSYM_DENSE5 (solid patterns) but has a more stringent definition. It is useful for generating gray text.

See the *OS/2 2.0 Presentation Manager Programming Reference* for definitions of these shading patterns.

ptlRefPoint Specifies the pattern origin for areas and thick lines. The pattern is mapped into the area to be filled by conceptually replicating the pattern definition in both the horizontal and vertical directions.

The pattern reference point is subject to all of the transforms. When an area is moved by changing a transform and redrawing, the fill pattern also appears to move so as to retain its position relative to the area boundaries. This allows part of a picture to be moved with a Bitblt operation and the remainder to be drawn by changing the appropriate transform with no discontinuity at the join.

The pattern reference point, which is specified in world coordinates, need not be inside the actual area to be filled and is not subject to clipping, although the area to be filled is subject to clipping.

adef The fields of an AREADEFS structure are:

defSet Area definition. This can be a text pattern, predefined pattern, bit map, pointer to an engine font, device font handle, or bit map handle.

fFlags The only valid flag is CDEF_GENERIC.

CodePage Code Page ID.

Character Attributes

The device character attributes are bundled in a DCHARBUNDLE structure:

Parameter	Description
cAttr	Size of the attribute bundle.
cDefs	Size of the CHARDEFS structure.
cbnd	CHARBUNDLE structure. See below.
cdef	CHARDEFS structure.

CHARBUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the CHARBUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
CBB_COLOR	lColor
CBB_BACK_COLOR	lBackColor
CBB_MIX_MODE	usMixMode
CBB_BACK_MIX_MODE	usBackMixMode
CBB_SET	usSet
CBB_MODE	usPrecision
CBB_BOX	sizfxCell
CBB_ANGLE	ptlAngle
CBB_SHEAR	ptlShear
CBB_DIRECTION	usDirection
CBB_TEXT_ALIGN	usTextAlign
CBB_EXTRA	fxExtra
CBB_BREAK_EXTRA	fxBreakExtra

cbnd The fields of a CHARBUNDLE structure are:

lColor Character foreground color.

lBackColor Character background color.

usMixMode Character foreground mix mode.

usBackMixMode Character background mix mode.

usSet Specifies a local identifier (lcid) for a logical font. If usSet is *0*, the current code page and character precision are used to resolve the selection of the base font. The code page (set by the function GreSetCodePage) identifies two base fonts, an outline font and an image font. The value of usPrecision determines which of these is selected. Valid values for usSet are:

0 Base font

Non-zero Local identifier for the logical font defined by the *cdef.defSet* field in the character attributes bundle.

usPrecision Specifies the character mode. The value of usPrecision is used to select output quality from Precision 1 (the lowest) to Precision 3. Presentation drivers normally use Precision 3 except when performance is improved by using the specified precision. For example, the EGA and VGA drivers always use Precision 3 when the current font is an outline font but switch to the specified precision for an image font.

Valid values for usPrecision are:

CM_MODE1 Precision 1. The selected font can be either an image or an outline font. When an image font is used, the first character is positioned with its reference point at the current position. Subsequent characters are positioned by using FONTMETRICS.

CM_MODE2 Precision 2. The selected font can be either an image or outline font. When an image font is used, the first character is positioned with its reference point at the current position. Subsequent characters are positioned by using the FONTMETRICS, CBB_BOX, CBB_ANGLE, and CBB_SHEAR attributes. This is done by constructing a transformation matrix

that transforms the FONTMETRICS values, sXDeviceRes and sYDeviceRes, to the rotated, scaled, and sheared character box attributes, and translates the reference point of the first character to the current position. The actual character positions are then determined by applying this transform to the character positions found by using the FONTMETRICS.

Notice that the character box attribute is subjected to the rotation, scaling, and shear defined by the current transformations, and defined by the other character attributes.

Note: Shear attributes affect only the horizontal position of CM_MODE2 characters when the character direction is CHRDIRN_TOPBOTTOM or CHRDIRN_BOTTOMTOP.

CM_MODE3 Precision 3. The selected font must be an outline font.

Note: CM_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

For outline fonts, regardless of mode, all character attributes together with the FONTMETRICS are used for positioning, scaling, rotating, and shearing the characters. This is done by constructing a transformation matrix as described above for CM_MODE2. The actual character vector stroke coordinates are then determined by applying this transform to the character-definition coordinates suitably modified (for character positioning) by the FONTMETRICS.

As with CM_MODE2, the character box attribute is subjected to the rotation, scaling, and shear defined by the current transformations, and defined by the other character attributes.

The *character reference point* is defined as the intersection of the base line and the left edge of the character. The *baseline* is defined as an offset, pCellOffset, from the top of the character cell.

sizfxCell Specifies fixed-point numbers for the width and height of a character cell in world-coordinate space. This defines the background area for a character. Each dimension is represented as a signed 4-byte integer with a notional binary point between bit 16 and bit 15. Therefore, +2.5 is represented by 00028000H and –2.5 is represented by FFFD8000H.

For CM_MODE1, the cell has no effect when characters are drawn from an image font. For CM_MODE2, the width determines the spacing of consecutive characters along the baseline. Both width and height can be positive, negative, or *0*. When either parameter is negative, the spacing occurs in the opposite direction to normal and each character is drawn reflected in CM_MODE3. For example, a negative height in the standard direction in Mode 3 indicates that the characters are drawn upside down, and that the string is drawn below the baseline (assuming no other transformations cause inversion). A zero character width or height is also valid. The string of characters collapses into a line. If both are *0*, the string is drawn as a single point in CM_MODE3.

ptlAngle Specifies integer values, x and y, for the coordinates of the end of a line starting at the origin (0, 0). The baseline for subsequent character strings is parallel to this line.

For CM_MODE1, the angle has no effect when characters are drawn.

For CM_MODE2, the angle is used to determine the position of each image character. However, the orientation of characters within the character box is inherent in their definitions. The characters are positioned so that the

lower-left corners of the character definitions are placed at the lower-left corners of the character boxes.

For CM_MODE3, the angle is observed accurately and the character boxes are rotated to be normal to the character baseline. If the coordinate system is such that one x-axis unit is not physically equal to one y-axis unit, a rotated character appears to be sheared.

ptlShear Specifies integer values, which identify the end coordinates of a line originating at 0, 0. The vertical strokes in subsequent outline character strings are drawn parallel to the defined line. For CM_MODE1, the shear has no effect when image characters are drawn. For CM_MODE2, the shear affects the height of the character cell. Therefore, the position of characters drawn with CDIRN_TOPBOTTOM or CDIRN_BOTTOMTOP. The top of the character box remains parallel to the character baseline.

If hx=0 and hy=1 (the standard default), *upright* characters result. If hx and hy are both positive or both negative, the characters slope from bottom left to top right. If hx and hy are of opposite signs, the characters slope from top left to bottom right. No character inversion takes place as a result of shear alone. (Inversion can be done with the charCell attribute.) Notice that it is incorrect to specify a zero value for hy because this would imply an infinite shear.

usDirection Valid values are:

CHDIRN_LEFTRIGHT	Left-to-right
CHDIRN_TOPBOTTOM	Top-to-bottom
CHDIRN_RIGHTLEFT	Right-to-left
CHDIRN_BOTTOMTOP	Bottom-to-top.

Note: CHDIRN_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

If the specified direction is not valid, the presentation driver uses CHDIRN_LEFTRIGHT as the default.

usTextAlign Specifies the horizontal and vertical alignment of character strings. This alignment defines a reference point within the string, which is positioned on the starting point specified for the string.

The horizontal alignment values are as follows:

TA_NORMAL_HORIZ Normal alignment. This is the initial default. The alignment assumed depends on the current character direction:

CHDIRN_LEFTRIGHT	Same as TA_LEFT
CHDIRN_TOPBOTTOM	Same as TA_CENTER
CHDIRN_RIGHTLEFT	Same as TA_RIGHT
CHDIRN_BOTTOMTOP	Same as TA_CENTER

TA_LEFT Left alignment. The string is aligned on the left edge of its leftmost character.

TA_CENTER Center alignment. The string is aligned on the arithmetic mean of Left and Right.

TA_RIGHT Right alignment. The string is aligned on the right edge of its rightmost character.

TA_STANDARD_HORIZ Standard alignment. The alignment assumed depends on the current character direction:

CHDIRN_LEFTRIGHT	Same as TA_LEFT

CHDIRN_TOPBOTTOM	Same as TA_LEFT
CHDIRN_RIGHTLEFT	Same as TA_RIGHT
CHDIRN_BOTTOMTOP	Same as TA_LEFT

The vertical alignment values are as follows:

TA_NORMAL_VERT Normal alignment. This is the initial default. The alignment assumed depends on the current character direction:

CHDIRN_LEFTRIGHT	Same as TA_BASE
CHDIRN_TOPBOTTOM	Same as TA_TOP
CHDIRN_RIGHTLEFT	Same as TA_BASE
CHDIRN_BOTTOMTOP	Same as TA_BOTTOM

TA_TOP Top alignment. The string is aligned on the top edge of its topmost character.

TA_HALF Half alignment. The string is aligned on the arithmetic mean of Bottom and Top.

TA_BASE Base alignment. The string is aligned on the base of its bottom character.

TA_BOTTOM Bottom alignment. The string is aligned on the bottom edge of its bottom character.

TA_STANDARD Standard alignment. The alignment assumed depends on the current character direction:

CHDIRN_LEFTRIGHT	Same as TA_BOTTOM
CHDIRN_TOPBOTTOM	Same as TA_TOP
CHDIRN_RIGHTLEFT	Same as TA_BOTTOM
CHDIRN_BOTTOMTOP	Same as TA_BOTTOM

The current position will also be modified relative to the current character direction as shown:

Horizontal For horizontal character directions:

Horizontal Alignment	**New X Current Position**
TA_LEFT	Right
TA_CENTER	Center
TA_RIGHT	Left

Vertical Alignment	**New X Current Position**
TA_TOP	Top
TA_HALF	Half
TA_BASE	Base
TA_BOTTOM	Bottom

Vertical For vertical character directions:

Horizontal Alignment	**New X Current Position**
TA_LEFT	Left
TA_CENTER	Center
TA_RIGHT	Right

Vertical Alignment	**New X Current Position**
TA_TOP	Bottom
TA_HALF	Half
TA_BASE	Base
TA_BOTTOM	Top

fxExtra A fixed point world-coordinate distance, which is added between every character as it is being placed.

fxBreakExtra A fixed point world-coordinate distance, which is added to the width of the break character as it is placed.

cdef The fields of a CHARDEFS structure are:

defSet Character set definition. If defSet is passed as *0*, the presentation driver must use the default device font (zero is passed only when the driver provides and manages its own default font). Otherwise, the significance of defSet depends upon the state of the CDEF_GENERIC flag. See below:

- If the flag is set, defSet is a pointer to an engine font.
- If not set, defSet is a device font identifier defined by the driver.

When defSet is a pointer to an engine font, cdef is a pointer to an instance of the FOCAFONT data structure. The definition of the FOCAFONT data structure is included in the header file. For a detailed description of the types used in the FOCAFONT data structure, refer to *Appendix E* in the *OS/2 2.0 Presentation Manager Programming Reference*.

fFlags Valid flags are:

CDEF_GENERIC	Engine font (not device font)
CDEF_BOLD	Font must be emboldened
CDEF_ITALIC	Font must be italicized
CDEF_UNDERLINE	Font must be underlined
CDEF_STRIKEOUT	Font must be StrikeOut.

CodePage Current code page. The presentation driver ignores this field when the font is not a multi-code page font that needs translating.

charSpacing Character spacing.

See the *OS/2 2.0 Programming Guide* for examples of these attributes.

Image Attributes

The device image attributes are bundled in a DIMAGEBUNDLE structure:

Parameter	Description
cAttr	Size of the attributes structure.
cDefs	Set to *0*. There is no IMAGEDEFS structure.
ibnd	IMAGEBUNDLE structure. See below.

IMAGEBUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the IMAGEBUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
IBB_COLOR	lColor
IBB_BACK_COLOR	lBackColor
IBB_MIX_MODE	usMixMode
IBB_BACK_MIX_MODE	usBackMixMode

ibnd The fields of an IMAGEBUNDLE structure are:

lColor	Image foreground color
lBackColor	Image background color
usMixMode	Image foreground mix mode
usBackMixMode	Image background mix mode.

Marker Attributes

The device marker attributes are bundled in a DMARKERBUNDLE structure:

Parameter	Description
cAttr	Size of MARKERBUNDLE structure
cDefs	Size of MARKERDEFS structure
mbnd	MARKERBUNDLE structure. See below.
mdef	MARKERDEFS structure. See below.

MARKERBUNDLE Mask: This mask is used in calls to GreDeviceSetAttributes to identify fields in the MARKERBUNDLE structure. Valid flags and the fields that they identify are:

Flag	Field
MBB_COLOR	lColor
MBB_BACK_COLOR	lBackColor
MBB_MIX_MODE	usMixMode
MBB_BACK_MIX_MODE	usBackMixMode
MBB_SET	usSet
MBB_SYMBOL	usSymbol
MBB_BOX	sizfxCell

mbnd The fields of a MARKERBUNDLE structure are:

lColor Marker foreground color.

lBackColor Marker background color.

usMixMode Marker foreground mix mode.

usBackMixMode Marker background mix mode.

usSet Specifies a local identifier (lcid) for the logical font:

0	Base marker set
Non-zero	Local identifier for the font identified in the *mdef.defSet* field of the marker attributes bundle.

usSymbol Specifies the identity of the required marker symbol in the current marker set. If the value of usSymbol does not identify a marker in the current font, the standard default for the font is used. The default for the default font is a cross. For a loaded font, it is the character identified by the usDefaultChar field of the FOCAMETRICS structure.

All values in the range 0 – 255 are valid. The defined values for the default marker set are:

MARKSYM_CROSS	Cross
MARKSYM_PLUS	Plus
MARKSYM_DIAMOND	Diamond
MARKSYM_SQUARE	Square
MARKSYM_SIXPOINTSTAR	Six-point star
MARKSYM_EIGHTPOINTSTAR	Eight-point star
MARKSYM_SOLIDDIAMOND	Filled diamond
MARKSYM_SOLIDSQUARE	Filled square
MARKSYM_DOT	Dot
MARKSYM_SMALLCIRCLE	Small circle
MARKSYM_BLANK	Blank.

Note: MARKSYM_DEFAULT is the default value at the API. It is a reserved value and is not passed to the presentation driver.

sizfxCell Specifies fixed-point numbers for the width and height of a marker cell in world-coordinate space. This defines the background area for a marker. Each dimension is represented as a signed 4-byte integer with a notional binary point between bit 16 and bit 15. Therefore, +2.5 is represented by 00028000H, and –2.5 is represented by FFFD8000H. The value of this attribute only affects the size of markers drawn with an outline (vector) font or marker set. Markers drawn from image (raster) sets are not affected.

mdef The fields of a MARKERDEFS structure are:

defSet Marker set definition. If this value is passed as zero, the presentation driver must use the default marker set. If the CDEF_GENERIC flag is set, this is a pointer to an engine font. Otherwise, it is a device-font identifier defined by the presentation driver.

fFlags Valid flags are:

CDEF_GENERIC	Engine font (not device font)
CDEF_BOLD	Marker must be emboldened
CDEF_ITALIC	Marker must be italicized
CDEF_UNDERLINE	Marker must be underlined
CDEF_STRIKEOUT	Marker must be StrikeOut.

CodePage Code page number.

Bit-Map Functions

Presentation drivers for hardcopy vector devices can return Failure on all bit-map operations. The same bit-map file format is used for bit maps, icons, and pointers. For details, refer to the *OS/2 2.0 Presentation Manager Programming Reference*.

Color Functions

By default, the color mode for a DC is set to index mode, and the DC has a Logical Color table set to the values given below. When in index mode, these defaults are always considered to be part of the color table unless they are explicitly overwritten by CreateLogColorTable (see page 8-34).

Note: Presentation drivers that support less than 16 colors must map Value 0 (CLR_BACKGROUND) through Value 15 (CLR_PALEGRAY) to device colors. If GreQueryColorData is called while the default color table is the current color table, the presentation driver returns the device colors.

Default values for the Logical Color table are:

CLR_FALSE (–5). All color planes or bits, or both, are FALSE.

CLR_TRUE (–4). All color planes or bits, or both, are TRUE.

CLR_DEFAULT (–3). This is the API default. It is a reserved value and is not passed to the presentation driver.

CLR_WHITE (–2). This index is never loaded explicitly. It always produces white when the default table is in force or when the index is set to RGB. When, with a realized color table and an index that is not RGB, this option is unavailable, it produces CLR_BACKGROUND.

CLR_BLACK (–1). This index is never loaded explicitly. It always produces black when the default table is in force or when the index is set to RGB. When, with a realized color table and an index that is not RGB, this option is unavailable, it produces CLR_NEUTRAL.

CLR_BACKGROUND (0). This is the natural background color of the device. For a hardcopy device, it is the paper color and for a display device it is the default window color, SYSCLR_WINDOW.

CLR_BLUE (1)

CLR_RED (2)

CLR_PINK (3)

CLR_GREEN (4)

CLR_CYAN (5)

CLR_YELLOW (6)

CLR_NEUTRAL (7). This is a device-dependent contrasting color. For a display device, it is the default window text color, SYSCLR_WINDOWTEXT.

CLR_DARKGRAY (8)

CLR_DARKBLUE (9)

CLR_DARKRED (10)

CLR_DARKPINK (11)

CLR_DARKGREEN (12)

CLR_DARKCYAN (13)

CLR_BROWN (14)

CLR_PALEGRAY (15)

Colors with indexes greater than *15* are device-dependent defaults, which must be defined by the presentation driver. The effective range of the color table which includes the default color table, is *–5* through *MaxIndex*. Color indexes outside this range that have not been loaded are not used by applications because these colors cannot be guaranteed.

Where physically possible, the default colors are always available on a device. For devices that support more than 16 colors, requested colors can be mapped to colors other than the defaults (when they exist). Such colors cannot be guaranteed to be similar for different devices. They can be different for other releases of applications and presentation drivers. Applications that depend on precise colors beyond the defaults must query the available colors (see "GreQueryRealColors" on page 8-123) and, when necessary, realize their own color tables (see "GreRealizeColorTable" on page 8-129).

Support for Monochrome Devices: Presentation drivers for monochrome devices must be able to draw pictures intended for color devices. A simple solution for hardcopy drivers is to map:

- Background color to paper color
- Foreground color to printer foreground except when:
  - In RGB mode. If the foreground RGB matches the default background RGB, use paper color.
  - In index mode. If the RGB foreground index matches the RGB color for Index 0, use paper color.

To map the RGB colors to the device, the presentation driver must first establish the *reset color*, which is the base color for the device. The reset color can be:

- Paper color for a hardcopy device with no loaded color table
- SYSCLR_WINDOW for a monochrome display with no loaded color table
- CLR_BACKGROUND for any monochrome device that has a loaded color table

Any color that is not the reset color is considered to be the *contrast color*. The values for the reset color and contrast color are either 000000H or FFFFFFH. When the reset color is 000000H, the contrast color is FFFFFFH. CLR_TRUE, CLR_FALSE, and CLR_DEFAULT are always honored independently of the reset color. The interpretation of CLR_BLACK and CLR_WHITE depends on the reset color.

When GreQueryNearestColor is called for a monochrome device, the value returned is either the reset color or the contrast color. GreErasePS causes the color to be set to the value of the reset color. See "GreErasePS" on page 8-62. GreBitblt can also be used to transfer a color bit map to a monochrome device or bit map. In this case, the source image background color becomes the reset color and all other pels are represented by the contrast color. See "GreBitblt" on page 8-26. More sophisticated presentation drivers for monochrome devices should use half-toning for colors to provide more usable output. Half-toning can be applied to all graphic primitives.

GreEscape

The GreEscape handling routine in the presentation driver supports the DevEscape function and its escape codes at the API. While the primary function of GreEscape is to implement the required support for the defined escape codes, it can be used to implement additional escape codes. There is a set of defined ranges for additional escape codes. The range chosen determines how the operating system processes the escape code when it is received as a parameter to DevEscape. On entry to GreEscape, the value of lEscape on the stack identifies the escape code. The action taken is determined by the escape code and the physical device that the presentation driver supports.

Support: GreEscape is called by DevEscape. GreEscape with the escape code, DEVESC_QUERYESCSUPPORT, must be supported by all presentation drivers. Hardcopy drivers also support the DEVESC_STARTDOC, DEVESC_ABORTDOC, DEVESC_NEXTFRAME, and DEVESC_ENDDOC escape codes. The other escape codes are optional. See "Defined Escape Codes" on page 8-16 and the individual escape codes that follow.

Stack Frame: On entry to the GreEscape routine, the stack frame contains:

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lEscape	LONG	Escape code.
clnCount	LONG	Number of bytes pointed to by plnData.
plnData	PBYTE	Pointer to input data structure.

Parameter	Data Type	Description
pcOutCount	PLONG	Pointer to the number of bytes in output data structure. If the escape code is one that returns data in the output data structure, the handling routine changes the value addressed by pcOutCount to show the number of bytes returned.
pOutData	PLONG	Pointer to output data structure.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape.

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Defined Escape Codes

The following list shows the escape codes that have been defined for Presentation Manager and the devices to which they apply. GreEscape returns DEVESC_NOTIMPLEMENTED for escape codes that it does not support.

DEVESC_ABORTDOC	(Hardcopy drivers only)
DEVESC_BREAK_EXTRA	(Hardcopy drivers only)
DEVESC_CHAR_EXTRA	(Hardcopy drivers only)
DEVESC_DBE_FIRST	(DBCS support)
DEVESC_DBE_LAST	(DBCS support)
DEVESC_DRAFTMODE	(Hardcopy drivers only)
DEVESC_ENDDOC	(Hardcopy drivers only)
DEVESC_FLUSHOUTPUT	(Hardcopy drivers only)
DEVESC_GETCP	(Hardcopy drivers only)
DEVESC_GETSCALINGFACTOR	(Hardcopy drivers only)
DEVESC_NEWFRAME	(Hardcopy drivers only)
DEVESC_NEXTBAND	(Hardcopy drivers only)
DEVESC_QUERYESCSUPPORT	(All drivers)
DEVESC_QUERYVIOCELLSIZES	(Display drivers only)
DEVESC_RAWDATA	(Hardcopy drivers only)
DEVESC_SETMODE	(Hardcopy drivers only)
DEVESC_STARTDOC	(Hardcopy drivers only)
DEVESC_STD_JOURNAL	(Hardcopy drivers only)

Ranges for Additional Escape Codes

The following table indicates the defined ranges for additional escape codes, and shows how the operating system processes the escape code when it is received as a parameter to DevEscape:

Table 8-1. Ranges for Additional Escape Codes

32768 – 40959	Escape is not metafiled or recorded by the spooler. The escape is passed to the presentation driver in all cases.
40960 – 49151	Escape is metafiled but is not recorded by the spooler. For an OD_QUEUED device with PM_Q_STD data and for all device types other than OD_METAFILE, the escape is passed to the presentation driver.
49152 – 57343	Escape is metafiled and recorded by the spooler. For an OD_METAFILE device or for OD_QUEUED with PM_Q_STD data, the escape is not passed to the presentation driver.
57344 – 65535	Escape is recorded by the spooler. The escape is passed to the presentation driver except when the DC is an OD_QUEUED device with PM_Q_STD data.

Line Functions

The style of a line determines whether it is drawn solid, alternating, invisible, or as any one of a combination of dots and dashes. This is determined by the usType parameter in the LINEBUNDLE structure (see "Line Attributes" on page 8-3), which can have any one of the ten values shown in Table 8-2. For each usType value, there is an associated 8-bit style mask whose bits form a template that corresponds to whether pels are set *on* or *off* when the line is drawn on the device.

The 8-bit style masks used by OS/2 2.0 are as follows:

Table 8-2.

usType	Value	StyleMask	Bits	Comment
LINETYPE_DEFAULT	0	0xFF	11111111	This is a solid line.
LINETYPE_DOT	1	0xAA	10101010	
LINETYPE_SHORTDASH	2	0xCC	11001100	
LINETYPE_DASHDOT	3	0xE4	11100100	
LINETYPE_DOUBLEDOT	4	0xA0	10100000	
LINETYPE_LONGDASH	5	0xE7	11100111	
LINETYPE_DASHDOUBLEDOT	6	0xEA	11101010	
LINETYPE_SOLID	7	0xFF	11111111	
LINETYPE_INVISIBLE	8	0x00	00000000	
LINETYPE_ALTERNATE	9	0xAA	10101010	

The style masks can be put into an array of bytes as illustrated in the following code example:

```
const BYTE bStyleMask[ ] =
{
 0xFF,0xAA,0xCC,0xE4,0xA0,0xE7,0xEA,0xFF,0x00,0xAA
};
```

Consider the following code example and comments:

```
DrawHorizontalLine(
  SHORT          sY,            /* Y-value
  SHORT          sX0,           /* Starting x-value
  SHORT          sX1,           /* Ending x-value
  USHORT         usStyleRatio,  /* Style ratio
  USHORT         usState        /* Current mask state
  PLINEBUNDLE    pLbnd)         /* pLbnd contains usType parameter
{
  USHORT         usStateOld;    /* Used for test
  USHORT         usOrigMask;    /* Original mask
  USHORT         usMask;        /* Current mask
```

The usState WORD structure is set up by the graphics engine. This structure is shown in Table 8-3.

Table 8-3. usState

5 bits (not used)	3 bits (mask position, value 0−7)	8 bits (error-term byte, value 0−0xFF)

Note: This format is different from the way the graphics engine stores this value. The graphics engine keeps the high and low bytes swapped, therefore, the calling routine must swap them before calling this function.

The mask position has three bits and eight possible values, 0−7, that correspond to the number of bits in the style mask. Notice that when the error-term byte overflows (exceeds 0xFF), it will increment the mask position value. To get the unshifted mask value:

```
usMask = usOrigMask = bStyleMasks[pLbnd -> usType];
```

The mask must be shifted to the current position as shown above:

```
usMask <<= ((usState & 0x700) >> 8);
```

For each pel:

```
for(sX = sX0;sX <= sX1;sX++){
```

Because many devices have small pel sizes, it is often necessary to set more than one pel *on* per corresponding bit in the style mask. This is accomplished by using a *style ratio*, which determines how many pels on the line correspond to a bit in the style mask. As an example, for each pel on the line, the most significant bit of the style mask is consulted. If this bit is *1*, the pel is drawn. If this bit is *0*, the pel is not drawn:

```
if(usMask & 0x80)
   DrawPel(sX,sY);

usStateOld = usState;
```

The style ratio is then added to the error-term byte of the usState parameter. When the error-term byte overflows, the mask position value (high byte of usState) is incremented to the next position. If the mask position exceeds *7*, it is reset to *0*:

```
usState = (usState + usStyleRatio) & 0x7ff;
```

The mask itself must now be shifted appropriately:

```
if(HIBYTE(usState) | = HIBYTE(usStateOld)){     /* Has it changed?                */
   if(HIBYTE(usState) == 0)                    /* Back to zero?                  */
      usMask - usOrigMask;                     /* Yes, reset it                  */
   else
      usMask <<= 1;                            /* Else shift it                  */
}
:
return(usState);                               /* Return the state for the next line */
```

Styled line information is usually maintained by the presentation driver in the Device Context (DC) instance data structure. See "Device Context" on page 1-8. Remember that the DC is a structure the presentation driver creates. The definition of the DC data structure is specific and can be unique for each unique presentation driver. This information can include:

```
typedef struc _DC {
:
USHORT      usStyleRatioX;    /* X-style ratio                                    */
USHORT      usStyleRatioY;    /* Y-style ratio                                    */
ULONG       lLineStyle;       /* Current state information (GRE Backward Format)  */
LINEBUNDLE  lbnd;             /* Linebundle information contains usType parameter */
BYTE        bMyCurrentMask;   /* Current style mask                               */
:
} DC;
typedef DC *PDC;
```

The lLineStyle value has much of the same information as the usState parameter in the example function. Its value is concurrently maintained by the graphics engine but has a different format from the example above. The graphics engine can set or query this value by calling the functions, GreSetLineOrigin or GreGetLineOrigin, respectively. Therefore, it is reasonable to keep a copy in the graphics engine format.

The graphics engine's format for this value consists of three values:

- A flag indicating whether or not to draw the first pel of the current line
- The current mask position (explained above)
- The error-term byte (explained above).

They are stored in the following manner:

High WORD 16-bits. Draw first pel flag. Can be 0x0000 (do not draw first pel) or 0x0001 (draw first pel).

Low WORD 16-bits, consisting of:

8 bits	Error-term byte
5 bits	GRE internal flags
3 bits	Mask position

If the two bytes of this low WORD are compared to the WORD in the above example, they are swapped.

Style Ratio: The most obvious way to generate styled lines is to draw a circle at the origin, and then draw a dashed line from the origin to all points on the circle. The number of dashes in each line will be the same and the lengths of all the dashes drawn will be equal. This is executed on firmware on some devices. However, but it is too costly in terms of CPU time to implement it in software. Therefore, an alternative method called the *maximum metric* is used in the graphics engine. This method can be visualized by drawing a square centered at the origin, and then drawing a dashed line to any point on the square. The number of dashes in each line will be the same but the lengths of the dashes will vary.

The maximum metric states that:

- If the line is y-major *as viewed on the device*:

  ```
  {y-major = ABS(y1—y0) > ABS(x1—x0)
  where (x0, y0) and (x1, y1) are the endpoints of the line in device-coordinate space},
  add pDC -> usStyleRatioY to error-term value upon each increment of y as line is drawn
  ```

- If the line is x-major *as viewed on the device*:

  ```
  {x-major = ABS(x1-x0) > ABS(y1-y0) in device-coordinate space},
  add pDC -> usStyleRatioX to error-term value upon each increment of x as line is drawn
  ```

The line style as viewed on the device is determined:

```
if (ABS(pDC -> usStyleRatioX * sDeltaX) > ABS(pDC -> usStyleRatioY * sDeltaY))
    LineStyleAsItLooksOnTheDevice = X-MAJOR;
else
    LineStyleAsItLooksOnTheDevice = Y-MAJOR;
```

This might be coded similar to the following:

```
if (ABS(pDC -> usStyleRatioX * sDeltaX) > ABS(pDC -> usStyleRatioY * sDeltaY)) {
    usChangeInStateForOnePixel = pDC -> usStyleRatioX; fAddThisWhen = X_INCREMENTS;
} else {
        usChangeInStateForOnePixel = pDC -> usStyleRatioY; fAddThisWhen = Y_INCREMENTS;
}
```

It follows that for a line (AB), the total change in style state can be expressed as:

```
usChangeInStateForWholeLine = MAX(ABS(pDC -> usStyleRatioX * sDeltaX),
                                  ABS(pDC -> usStyleRatioY * sDeltaY))
where sDeltaX = Bx-Ax and sDeltaY = By-Ay
```

For example, an EGA device that has a 640x350 (x-to-y) resolution is displayed on a monitor, which has an x-to-y ratio of 1-to-.75, respectively. To calculate the aspect ratio:

```
x/y Ratio = 350/(640*.75) = .72917   Therefore:  x = y*.72917 or y = x/.72917
This indicates that a pel is taller than it is wide.
```

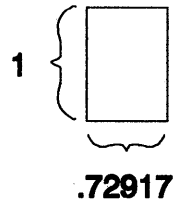


Figure 8-1. Pel

Assume that four pels in the x-direction is the desirable size of a styled-line dot. Because this display is 9.5 inches across and there are 640 pels across, the length of the four pels is:

```
(9.5inches/640pels) * 4pels = .059375inches
This results in a pDC -> usStyleRatioX = 64:
                                  64 = 256/4
```

Notice that 256 = 0x100, which corresponds to an overflow of the error-term byte into the mask position. To get the equivalent pDC -> usStyleRatioY value, take the desired distance and multiply it by *y* pels per inch:

```
(.059375inches * 350pels)/7.125inches = 2.917pels
Therefore: pDC -> usStyleRatioY = 256/2.917 = 87.76 (rounded to 88)
```

An easier method is to calculate pDC -> usStyleRatioY from pDC -> usStyleRatioX using the aspect ratio:

```
pDC -> usStyleRatioY = pDC -> usStyleRatioX / .72917 = 87.76 (rounded to 88)
```

Notice that the values, pDC -> usStyleRatioX and pDC -> usStyleRatioY, are the same as those returned by GreGetStyleRatio.

An example of what is meant by *as viewed on the device* is as follows: If a line is drawn from (0, 0) to (100, 100) pels on the device, it is drawn as a diagonal line but does not look diagonal to an observer. Instead, it looks like a line drawn to (100, 73). This is because of the *aspect ratio*. Each unit (pel) in the x-direction travels only .72917 as far as a unit that travels in the y-direction. All of the images on the device do not look skewed because this is factored in when the application draws a diagonal line (for example, by drawing from (0, 0) to (73, 100)). The styled lines are affected when a line that is drawn in pels as x-major *appears* on the device as y-major. In this case, a line drawn from (0, 0) to (85, 64) is x-major as drawn in

pels (because ABS(x1-x0) > ABS(y1-y0)) but *appears* on the device to be y-major. Notice that this line *must* be styled y-major to look right on the device:

```
usChangeInStateForOnePixel = ABS(pDC -> usStyleRatioX * sDeltaX) > ABS(pDC -> usStyleRatioY * sDeltaY)?
                             pDC -> usStyleRatioX : pDC -> usStyleRatioY;
where:
     ABS(pDC -> usStyleRatioX * sDeltaX) = 64 * 85 = 5440
     ABS(pDC -> usStyleRatioY * sDeltaY) = 88 * 64 = 5632
```

Because 5440 < 5632, the line is styled y-major by using pDC -> usStyleRatioY. The x-major diagonal line drawing routine usually adds pDC -> usStyleRatioX to the error-term byte for every increment of x. This is correct if the aspect ratio is 1:1. However, because the aspect ratio is not 1:1, pDC -> usStyleRatioY must be used in this case for every increment of y *although the line is drawn as an x-major line with the x-major routine*. This means that regardless of which way the line is drawn, it must be styled according to how it looks on the device, which is determined by the maximum metric method described above.

LINETYPE_ALTERNATE is a special case of styled line. When drawing a line of this type (pLbnd -> usType = LINETYPE_ALTERNATE), the x and y style ratios are *temporarily* set to 256 to set every other pel on the line *on*. Notice that changing the values returned by GreGetStyleRatio is not necessary because the graphics engine does not call this function if the line type is LINETYPE_ALTERNATE.

PolyShortLines and Styling: The graphics engine determines how to style the PolyShortLine, and either sets the PSL_YMAJOR bit of the style field or clears it to *0*. Therefore:

```
if(psl -> usStyle & PSL_YMAJOR) {

Style it y-major by adding pDC -> usStyleRatioY to the error-term value upon each increment of y
as it is drawn.
}
else {

Style it x-major by adding pDC -> usStyleRatioX to the error-term value upon each increment of x
as it is drawn.
}
```

First and Last Pel Considerations: It is the responsibility of the presentation driver to ensure that a series of line, arc, and fillet orders all join up correctly including the on/off counts defined by the current line attributes. For example, when drawing connected lines (PolyLines), the handling routine must not draw the first pel of the second, and subsequent, lines. Typically, the presentation driver maintains a flag in the DC instance data structure to indicate whether the first pel of a line is to be drawn. This flag is set by GreSetCurrentPosition and cleared by any subsequent drawing primitive. To ensure that a figure is closed correctly, GreCloseFigure does not draw the last pel in the closure line.

Some orders are defined as *move* type operations. A move causes three things to happen:

- Line style sequence is reset.
- The next line, arc, fillet, or partial arc primitive is drawn with first and last pel (subject to the line style sequence).
- In an area, if the current figure is not closed (that is, the current device coordinate position is not the same as the start device coordinate position), an implicit closure line is drawn to close it.

Subsequent start line, arc, fillet, and partial arc primitives are drawn to include the last but not the first pel (subject to the line style sequence). Any closed figure (full arc, box, or pie slice drawn with a boundary), is drawn with its boundary complete (no missing pels) and with the line-pattern sequence honored around all the parts of its boundary. Such closed figures are not considered to be move type operations, and allow construction of complex area boundaries.

Move type operations are:

- GreSetCurrentPosition.
- Any GreSet*xxx* function that changes or might change the transform from world-coordinate space to device coordinates. For example, GreSetModelTransform or GreSetWindow/ViewportTransform.
- Any GreSet*xxx* function that changes or might change the current clipping. For example, GreSetViewingLimits.

A different set of rules is necessary to construct a boundary for scan-line area filling. For example, ignore line style and draw all lines solid with first pel *off*, last pel *on* (see "GreGetLineOrigin" on page 8-90). This boundary is different from the boundary that is drawn on the screen after the interior is filled. Functions such as GrePolyLine, GreArc, and GrePolyFillet that are preceded by a move operation are drawn with the first pel *on* and the last pel set *off*.

Mandatory Functions (for All Drivers) by Category

Related mandatory functions for all presentation drivers can be grouped together into the following categories:

Attribute Functions

- GreDeviceGetAttributes (see page 8-43)
- GreDeviceSetAttributes (see page 8-48)
- GreDeviceSetGlobalAttribute (see page 8-51)
- GreGetPairKerningTable (see page 8-91).

Bit-Map Functions

- GreBitblt (see page 8-26)
- GreDeviceCreateBitmap (see page 8-36)
- GreDeviceDeleteBitmap (see page 8-41)
- GreDeviceSelectBitmap (see page 8-47)
- GreDrawBits (see page 8-53)
- GreDrawBorder (see page 8-57)
- GreGetBitmapBits (see page 8-83)
- GreGetPel (see page 8-92)
- GreImageData (see page 8-93)
- GreSetBitmapBits (see page 8-134)
- GreSetPel (see page 8-142).

Color Table Functions

- GreCreateLogColorTable (see page 8-34)
- GreQueryColorData (see page 8-108)
- GreQueryColorIndex (see page 8-109)
- GreQueryLogColorTable (see page 8-120)
- GreQueryNearestColor (see page 8-121)
- GreQueryRealColors (see page 8-123)
- GreQueryRGBColor (see page 8-125).
- GreRealizeColorTable (see page 8-129).
- GreUnrealizeColorTable (see page 8-144).

Device Functions 2

- GreDeviceQueryFontAttributes (see page 8-44)
- GreDeviceQueryFonts (see page 8-45)
- GreErasePS (see page 8-62)
- GreNotifyClipChange (see page 8-96)
- GreNotifyTransformChange (see page 8-97)
- GreRealizeFont (see page 8-130).

Device Functions 3

- GreAccumulateBounds (see page 8-25)
- GreDeviceSetDCOrigin (see page 8-50)
- GreGetBoundsData (see page 8-86)
- GreGetCodePage (see page 8-87)
- GreGetDCOrigin (see page 8-89)
- GreGetLineOrigin (see page 8-90)
- GreLockDevice (see page 8-95)
- GreResetBounds (see page 8-133)
- GreSetCodePage (see page 8-137)
- GreSetLineOrigin (see page 8-140)
- GreUnlockDevice (see page 8-143).

GreEscape Functions

- GreEscape DEVESC_ABORTDOC (see page 8-63)
- GreEscape DEVESC_BREAK_EXTRA (see page 8-65)
- GreEscape DEVESC_CHAR_EXTRA (see page 8-66)
- GreEscape DEVESC_DBE_FIRST (see page 8-67)
- GreEscape DEVESC_DBE_LAST (see page 8-68)
- GreEscape DEVESC_DRAFTMODE (see page 8-69)
- GreEscape DEVESC_ENDDOC (see page 8-70)
- GreEscape DEVESC_FLUSHOUTPUT (see page 8-71)
- GreEscape DEVESC_GETCP (see page 8-72)
- GreEscape DEVESC_GETSCALINGFACTOR (see page 8-73)
- GreEscape DEVESC_NEWFRAME (see page 8-74)
- GreEscape DEVESC_NEXTBAND (see page 8-75)
- GreEscape DEVESC_QUERYESCSUPPORT (see page 8-76)
- GreEscape DEVESC_QUERYVIOCELLSIZES (see page 8-77)
- GreEscape DEVESC_RAWDATA (see page 8-79)
- GreEscape DEVESC_SETMODE (see page 8-80)
- GreEscape DEVESC_STARTDOC (see page 8-81)
- GreEscape DEVESC_STD_JOURNAL (see page 8-82).

Line Functions

- GreDisjointLines (see page 8-52)
- GreDrawLinesInPath (see page 8-60)
- GreGetCurrentPosition (see page 8-88)
- GrePolyLine (see page 8-99)
- GrePolyScanline (see page 8-102)
- GrePolyShortLine (see page 8-104)
- GreSetCurrentPosition (see page 8-138).

Marker Function

- GrePolyMarker (see page 8-101).

Query Functions

- GreQueryDeviceBitmaps (see page 8-110)
- GreQueryDeviceCaps (see page 8-111)
- GreQueryDevResource (see page 8-113)
- GreQueryHardcopyCaps (see page 8-118).

Text Functions

- GreCharString (see page 8-30)
- GreCharStringPos (see page 8-31)
- GreQueryCharPositions (see page 8-106)
- GreQueryTextBox (see page 8-126)
- GreQueryWidthTable (see page 8-128).

GreAccumulateBounds

```
#define INCL_GRE_DEVMISC3

BOOL GreAccumulateBounds (hdc, prclRect, pInstance, lFunction)
```

This function is called to merge bounds into the total bounds held by the presentation driver. The presentation driver does bounds calculations for all drawing primitives. It must convert the bounds to model space as they are accumulated before merging with the GPI bounds. This can be done with GreConvert. GreAccumulateBounds is related to GreResetBounds (page 8-133) and GreGetBoundsData (page 8-86).

Support: This function must be supported by the presentation driver. GreAccumulateBounds is used when a drawing is created to maintain a rectangle that forms the bounding box for the entire drawing. This rectangle is used in transforms and other functions that manipulate the entire drawing at once. GreAccumulateBounds can be handled by bit-map emulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
prclRect	PRECTL	Pointer to rectangle, defined as a RECTL structure in device coordinates
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreAccumulateBounds

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreBitblt

```
#define INCL_GRE_BITMAPS

LONG GreBitblt (hdc, hdcSrc, cPoints, paptlPoint, lRop, flOptions, pBattrs, pInstance, lFunction)
```

This function modifies bit-map data at a target rectangle in the current DC. The modification can copy a rectangle of data from a specified source DC to the target or perform a raster operation on the target. The device contexts can be memory DCs with bit maps selected, or DCs belonging to devices that support raster operations.

When copying bits from a color bit map to a monochrome bit map or device, only those pels that are in the source background color are copied to the target as background color. All other pels are copied to the target as foreground color. Copying is nondestructive. When the target and source rectangles are in the same DC, no information is lost from the source if the rectangles overlap. When the target is expressed in world coordinates (that is, the BBO_TARGWORLD flag is set in flOptions), they must be transformed to device coordinates. The bits are transferred to an upright rectangle in device space, regardless of any rotational elements that might have been present in the transforms.

The attribute structure identified by the pBattrs parameter defines the bit-map foreground and background colors. If pBattrs is NULL, the handling routine uses the current foreground and background colors.

When the mix specified by lRop requires both source and pattern, a 3-way operation is performed by using the current pattern in the target DC. If pattern mixing is not required, a 2-way operation is done. If any of the source data is unavailable, the handling routine transfers those bits that are present and returns without error. This might occur when the source DC is a window on the screen that has been overlaid by another. In this example, the handling routine must proceed by reading what is there.

Support: This function must be supported by the presentation driver. GreBitblt is called by the function GpiBitBlt, and is used to modify bit-map data within a target rectangle of the current device context. However, if the destination is larger or smaller than the source, the presentation driver can pass this function to the graphics engine by using the original pointer copied from the dispatch table.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hdcSrc	HDC	Source device context or bit-map handle.
cPoints	LONG	Number of (x, y) pairs in paptlPoint. See below.
paptlPoint	PPOINTL	Pointer to an array of (x, y) coordinate pairs. See below.
lRop	LONG	Raster operation code. See below.
flOptions	ULONG	Specifies treatment of eliminated lines and columns when compression is done. See below.
pBattrs	PBITBLTATTRS	Pointer to attributes. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreBitblt.

hdcSrc Handle to the source DC or bit map. When lRop does not require a source, hdcSrc is passed as NULL. The handling routine then copies the current pattern to the currently selected bit map or device.

cPoints A count of the number of (x, y) pairs in the paptlPoint array. The count can be 2, 3, 4:

cPoints = 2 A raster operation (as determined by lRop) on the destination rectangle.

cPoints = 3 A copy between two rectangles of the same size. Only the bottom-left corner is given for the source rectangle.

cPoints = 4 Operation is determined by comparing the sizes of the two rectangles:

Target<Source Compress the source rectangle into the target rectangle. The flOptions flags determine how to handle eliminated rows and columns. In this case, the function can be passed to the GreBitblt routine in the graphics engine.

Target = Source Copy between equal rectangles.

Target>Source Stretch the source rectangle into the target rectangle. In this case, the function can be passed to the GreBitblt routine in the graphics engine.

paptlPoint Pointer to a block of (x, y) coordinate pairs that define the target and source rectangles. The coordinates, which can be passed as a pair of RECTL structures, define the bottom-left and top-right corners of the target and source rectangles (see cPoints above for exceptions):

```
(xTgtBL, yTgtBL), (xTgtTR, yTgtTR), (xSrcBL, ySrcBL), (xSrcTR, ySrcTR)
```

When the source rectangle is totally or partially outside the source bit map (or device), the operation is implementation-dependent for that area (that is, the programmer of the called presentation driver must decide what to do).

Note: When BBO_TARGWORLD is not set, the rectangles are noninclusive. That is, they include the left and lower boundaries in device units but not the top and right boundaries. When the bottom-left corner of a rectangle maps to the same device pel as the top-right corner, that rectangle is considered to be empty.

When BBO_TARGWORLD is set, the target rectangle is inclusive at all boundaries. The source is noninclusive.

lRop Raster operation code. The low-order byte represents a mix value in the range 00H – FFH. Raster operation code values and the mix-bit table are defined in the *OS/2 2.0 Presentation Manager Programming Reference*. The handling routine uses lRop to determine the operations to perform on the pattern, source, and target to get the required mix.

In addition to the ROP values defined at the API, the presentation driver must support ROP_GRAY (000080CAH). This value is used to shade the text for menu items that are not currently selectable. When ROP_GRAY is set, the handling routine overpaints the foreground pattern by using the current pattern and the background pattern color (background pels for the pattern are not changed). For the PATSYM_HALFTONE pattern, this overpaints the background pattern color onto alternate pels leaving those in between unchanged.

flOptions Option flags:

BBO_OR Stretch and compress, as necessary, ORing any eliminated rows and columns. Used for White on Black.

BBO_AND Stretch and compress, as necessary, ANDing any eliminated rows and columns. Used for Black on White.

BBO_IGNORE Stretch and compress, as necessary, ignoring any eliminated rows and columns. Used for color.

BBO_TARGWORLD The target rectangle is defined in world coordinates in the target PS. When this option is specified, the target rectangle is transformed to device coordinates. Where any shear or rotation has occurred, this must be converted to an upright rectangle that

bounds the transformed figure. This is then used as the target for the operation. No inversion of the image takes place.

BLTMODE_SRC_BITMAP hdcSrc is a bit-map handle. The bit map must not be currently selected into a device context. If this flag is not set, hdcSrc is a DC handle.

BLTMODE_ATTRS_PRES If set, the pBattrs parameter is present. This option can be ORed with any of the above options.

Note: Flags 15–31 are not used by the system. They are reserved for use by the presentation driver.

pBattrs This points to a BITBLTATTRS structure:

cSize Size of this structure
lColor Foreground color of source
lBackColor Background color of source.

The color values are used in conversions between monochrome and color data, and is the only format conversion required. The conversions are required for:

- Output of a monochrome pattern to a color device. In this case, the source pattern is converted to a color pattern. This is performed by using the colors provided in the BITBLTATTRS structure. If these colors are not provided, the handling routine uses the current area colors for the target DC. See "Area (Pattern) Attributes" on page 8-5. The bits are then transferred so that:
  - Source 1s become (target area) foreground color
  - Source 0s become (target area) background color.
- Transfer from a monochrome bit map to a color bit map or device. In this case, the source bits are converted by using the current image colors. These are the colors provided in the BITBLTATTRS structure. If these colors are not provided, the handling routine uses the current image colors for the target DC. See "Image Attributes" on page 8-11. The bits are then transferred so that:
  - Source 1s become (target image) foreground color
  - Source 0s become (target image) background color.
- Transfer from a color bit map to a monochrome bit map or device. In this case, the source bit map is converted by using the source and target image colors. The target colors are provided in the BITBLTATTRS structure. If these colors are not provided, the handling routine uses those in the image attributes bundle for the target DC. See "Image Attributes" on page 8-11. When the source is a device context, the source-image background color is that from the source DC. When the source is a bit-map handle, the background color is taken from the BITBLTATTRS structure, if provided, or otherwise from the background-image color of the target DC. The bits are then transferred so that:
  - Source pels that are the source-image background color become target-image background color.
  - All other pels become target-image foreground color.

When lRop does not call for a pattern, the pattern set and pattern symbol are not used. Neither the source nor the pattern is required when a bit map or part of a bit map is being cleared to a particular color. When a pattern is required, dithering can be done for solid patterns in a color that is not available on the device. Color dithering is described on page 8-121.

Return Codes: On completion, the handling routine must return an LONG integer (cHits), indicating, where appropriate, whether correlation hits have been detected:

GPI_OK	Successful
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INCOMPATIBLE_BITMAP
PMERR_INCORRECT_DC_TYPE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_BITBLT_MIX
PMERR_INV_BITBLT_STYLE
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_INDEX
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_DC_DATA
PMERR_INV_DC_TYPE
PMERR_INV_DRIVER_NAME
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_SCAN_START
PMERR_INV_USAGE_PARM
PMERR_UNSUPPORTED_ATTR_VALUE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCharString

```
#define INCL_GRE_STRINGS

LONG GreCharString (hdc, cChars, pchString, pInstance, lFunction)
```

This function draws a character string starting at the current (x, y) position. Upon completion, the current (x, y) position is the start point for the character cell immediately after the last character in the string.

Support: GreCharString must be supported by the presentation driver. The handling routine must provide full support for drawing characters from an image font in CM_MODE1 when the character direction is CHDIRN_LEFTRIGHT (see "Character Attributes" on page 8-6). For outline characters or characters in any other mode or direction, the handling routine can dispatch the call to the graphics engine at the address given for this call in the default dispatch table.

GreCharString is called by the function GpiCharString. GreCharString is used to draw a character string from the current position within the presentation space. It updates the current presentation space position upon completion of output and produces a call to GreSetCurrentPosition.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
cChars	LONG	Number of characters in string
pchString	PCH	Pointer to character string
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCharString

Return Codes: On completion, the handling routine must return a LONG value (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_FONT_AND_MODE_MISMATCH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_LENGTH_OR_COUNT
PMERR_PATH_LIMIT_EXCEEDED.

GreCharStringPos

```
#define INCL_GRE_STRINGS

LONG GreCharStringPos (hdc, pptlStart, prclRect, flOptions, cChars, pchString, pAdx, pAttrs, pInstance, lFunction)
```

This function draws a character string. The string can be drawn from the current (x, y) position or from a position specified.

Support: GreCharStringPos must be supported by the presentation driver. The handling routine must provide full support for drawing characters from an image font in CM_MODE1 when the character direction is CHDIRN_LEFTRIGHT (see "Character Attributes" on page 8-6). For outline characters or characters in any other mode or direction, the handling routine can dispatch the call to the graphics engine at the address given for this call in the default dispatch table.

GreCharStringPos is called by the function GpiCharStringAt. GreCharStringPos is used to draw a character string either at the current position or at a specified position. It will also update the current presentation space position upon completion of output.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pptlStart	PPOINTL	Pointer to (x, y) coordinates of start position.
prclRect	PRECTL	Pointer to an opaque or clip rectangle. See below.
flOptions	ULONG	Flags. See below.
cChars	LONG	Number of characters in string.
pchString	PCH	Pointer to character string.
pAdx	PLONG	Pointer to Increment array. See below.
pAttrs	PCSP_INFO	Pointer to attributes structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCharStringPos.

prclRect The clipping rectangle pointed to by this parameter is defined as a RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

This rectangle, which is in world coordinates, is used as the clipping rectangle or as the background for the string (or both) depending on the value of flOptions. When the CHS_OPAQUE flag is set, normal background mix attributes are ignored and the rectangle is drawn using overpaint and the character background color attribute. When the CHS_OPAQUE is not set, the background is drawn using the normal method. When neither CHS_OPAQUE nor CHS_CLIP are specified, this parameter is ignored. Notice that points on the boundary of this rectangle are considered to be inside the rectangle.

flOptions The following flags can be used in combination:

CHS_OPAQUE	Background of characters is defined by the rectangle prclRect. The rectangle is to be shaded (with background color and overpaint) before drawing.
CHS_VECTOR	Increment vector supplied (pAdx). If *0*, pAdx is ignored.
CHS_LEAVEPOS	Leave current position at the start of string.
CHS_CLIP	Clip string to rectangle.
CHS_START_XY	Start position of the string. When set, the handling routine must draw the string from the position indicated by pptlStart. If this flag is not set, the current position is used.
CHS_ATTR_INFO	Attributes to be used. When this flag is set, pAttrs indicates the foreground and background colors. Current attributes are unchanged. If the flag is not set, the string is drawn using the current character attributes. See "Character Attributes" on page 8-6.
CHS_UNDERSCORE	Underscore the characters. See the FATTRS structure in "GreCreateLogicalFont" on page 11-14.
CHS_STRIKEOUT	Overstrike the characters.

pAdx Pointer to an array of LONG integers, one element for each character in the string. When CHS_VECTOR is set, this array is used to set the spacing between characters. Each element is the distance in world coordinates from the bottom-left corner of the corresponding character in the string to the bottom-left corner of the next. The distance is measured along the baseline for left-to-right and right-to-left character directions, and along the shear line for top-to-bottom and bottom-to-top character directions. The final element is used to reposition the current position, when necessary.

pAttrs Pointer to a CSP_INFO structure. This structure contains the attributes to be used to draw the string when the CHS_ATTR_INFO flag is set. These do not alter the current character attributes (see "Character Attributes" on page 8-6). The CSP_INFO structure is defined as:

cSize	Number of bytes in structure
lColor	Use foreground color
lBackColor	Use background color.

Return Codes: On completion, the handling routine must return a LONG value (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_FONT_AND_MODE_MISMATCH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY

PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_LENGTH_OR_COUNT
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCreateLogColorTable

```
#define INCL_GRE_COLORTABLE

BOOL GreCreateLogColorTable (hdc, flOptions, lFormat, lStart, cCount, pData, pInstance, lFunction)
```

This function defines the entries of the logical color table.

Support: This function must be supported by the presentation driver. GreCreateLogColorTable is called by GpiCreateLogColorTable to create a logical color table, which is used in subsequent drawing operations.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	See below.
lFormat	LONG	Format of entries in the table. See below.
lStart	LONG	Starting index, only relevant for LCOLF_CONSECRGB.
cCount	LONG	Number of elements supplied in application data area. See below.
pData	PLONG	Pointer to application data area. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCreateLogColorTable.

flOptions Valid options are:

LCOL_RESET Indicates that the handling routine must reset the color table to default before processing the remainder of this function.

Note: This option is assumed when the color table is changed from LCOLF_RGB to LCOLF_INDRGB or LCOLF_CONSECRGB.

LCOL_REALIZABLE Indicates that the application can call GreRealizeColorTable at the appropriate time. This can affect the way the handling routine maps the indexes when the logical color table is not realized. A realizable color table is only required to provide color mapping for its color indexes while it is realized.

If this flag is not set, GreRealizeColorTable has no effect and posts a warning.

LCOL_PURECOLOR For solid patterns, pattern colors that are not available can be approximated by dithering. When this flag is set, only pure colors are used; the handling routine must not dither colors. The default is to allow color dithering.

Other flags are reserved and must be *0*.

lFormat Valid formats are:

LCOLF_INDRGB Array of (index, RGB) values. Each pair of values contains 8 bytes, a 4-byte index and a 4-byte color. This sets the color table into index mode, and forces LCOL_RESET if it is in RGB mode.

LCOLF_CONSECRGB Array of (RGB) values corresponding to color indexes starting from lStart upwards. Each entry is a 4-byte value. This sets the color table into index mode, and forces LCOL_RESET if it is in RGB mode.

LCOLF_RGB Color index = RGB. This sets the color table to RGB mode.

cCount The number of elements supplied in pData. This can be set to *0* if the color table is to be reset to the default, or LCOLF_RGB. When this is *0*, LCOLF_INDRGB and LCOLF_CONSECRGB have the same effect.

For LCOLF_INDRGB, cCount must be an even number.

pData Data area containing the color table definition data. The format depends on the value of lFormat. Each color value is a 4-byte integer with a value of:

```
(R*65536) + (G*256) + B
```

where:

```
R=red intensity value
G=green intensity value
B=blue intensity value
```

The maximum intensity for each primary is 255.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: Error checking for this function is performed by the graphics engine. Error codes for conditions the handling routine can expect to be passed by the graphics engine include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_INDEX
PMERR_INV_COLOR_START_INDEX
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_REALIZE_NOT_SUPPORTED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: See GpiCreateLogColorTable in the *OS/2 2.0 Presentation Manager Programming Reference* for a full description of this function.

GreDeviceCreateBitmap

```
#define INCL_GRE_BITMAPS

ULONG GreDeviceCreateBitmap (hdc, pInfoHd, flUsage, pBitmap, pInfo, pInstance, lFunction)
```

This function creates a bit map and obtains its handle.

Support: This function must be supported by the presentation driver. GreDeviceCreateBitmap is called from GreCreateBitmap, which is one of the graphics engine internal device support functions.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pInfoHd	PBITMAPINFOHEADER	Pointer to BITMAPINFOHEADER or BITMAPINFOHEADER2 structure defining the new bit map. See below.
flUsage	ULONG	Additional information used when creating a new bit map. See below.
pBitmap	PBYTE	Pointer to bit-map initialization data. See below.
pInfo	PBITMAPINFO	Pointer to BITMAPINFO or BITMAPINFO2 structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceCreateBitmap.

pInfoHd Pointer to either a BITMAPINFOHEADER structure:

cbFix Length in bytes of this structure
cx Bit-map width
cy Bit-map height
cPlanes Number of color planes, *1* if standard format
cBitCount Number of adjacent color bits per pel.

Notice that each plane has ((cx*cBitCount+31)/32*4*cy) bytes.

Or pointer to a BITMAPINFOHEADER2 structure:

cbFix Length in bytes of this structure

cx Bit-map width

cy Bit-map height

cPlanes Number of color planes, *1* if standard format

cBitCount Number of adjacent color bits per pel

ulCompression Compression scheme used to store the bit map:

BCA_UNCOMP Bit map is uncompressed (the only valid value).

cbImage Length of bit-map storage data in bytes. If the bit map is uncompressed, *0* (default) can be specified for this.

cxResolution Horizontal component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution Vertical component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cclrUsed The number of color indexes from the color table that are used by the bit map. If it is *0* (default), all the indexes are used. If it is non-zero, only the first cclrUsed entries in the table are accessed by the system. Further entries can be omitted.

For standard formats with a cBitCount of 1, 4, or 8 (and cPlanes = 1), any indexes beyond cclrUsed are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, cclrUsed is the number of colors used by the bit map.

cclrImportant Minimum number of colors indexes for satisfactory appearance of the bit map. More colors can be used in the bit map, however, it is not necessary to assign them to the device palette. These additional colors can be mapped to the nearest colors available. Zero (default) means that all entries are important. For a 24-bitcount standard format, the cclrImportant colors are also listed in the color table relating to this bit map.

usUnits Units of measure of the horizontal and vertical components of resolution:

BRU_METRIC (Default.) Pels per meter.

usReserved Reserved field. If present, it must be *0*.

usRecording Recording algorithm, the format in which bit-map data is recorded:

BRA_BOTTOMUP (Default.) Scan lines are recorded from bottom-to-top.

usRendering Half-toning algorithm used to record bit-map data that has been digitally half-toned:

BRH_NOTHALFTONED (Default.) Bit-map data not half-toned.

BRH_ERRORDIFFUSION Error diffusion or damped error diffusion algorithm.

BRH_PANDA Processing algorithm for noncoded document acquisition.

BRH_SUPERCIRCLE Super circle algorithm.

cSize1 Size value *1*. If BRH_ERRORDIFFUSION is specified in usRendering, cSize1 is the error damping as a percentage in the range *0—100*. A value of 100% indicates no damping. A value of 0% indicates that any errors are not diffused.

If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize1 is the x-dimension of the pattern used in pels.

cSize2 Size value *2*. If BRH_ERRORDIFFUSION is specified in usRendering, this parameter is ignored. If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize2 is the y-dimension of the pattern used in pels.

ulColorEncoding Color encoding:

BCE_RGB (Default.) Each element in the color array is an RGB2 data type.

ulIdentifier Reserved for application use.

flUsage The only valid flag is:

CBM_INIT When set, the pBitmap and paInfo parameters are used to initialize the newly created bit map. It is assumed that enough data is passed to initialize the whole bit map.

Other flags are reserved and must be ignored by the handling routine.

pBitmap Pointer to the pel data of the bit map. This data is stored in the order that the coordinates appear on a display screen, that is, the pel in the lower-left corner is the first in the bit map. Pels are scanned to the right, and upward, from that position. The bits of the first pel are stored beginning with the most significant bits of the first byte. The data for pels in each scan line is packed together tightly. However, all scan lines are padded at the end so that each one begins on a ULONG boundary. That is, three bytes of pel data will hold one 24-bit pel, three 8-bit pels, six 4-bit pels, or twenty-four 1-bit pels. If those three bytes are the only pel data for that scan line, one more byte of zeros would be required to pad the line to a ULONG boundary.

pInfo Pointer to either a BITMAPINFO structure:

cbFix Length of structure
cx Bit-map width
cy Bit-map height
cPlanes Number of color planes, *1* if standard format
cBitCount Number of adjacent color bits per pel
argbColor[] Color table array of RGB structures:
bBlue
bGreen
bRed

Or pointer to a BITMAPINFO2 structure:

cbFix Length of structure

cx Bit-map width

cy Bit-map height

cPlanes Number of color planes, *1* if standard format

cBitCount Number of adjacent color bits per pel

ulCompression Compression scheme used to store the bit map:

BCA_UNCOMP Bit map is uncompressed (the only valid value).

cbImage Length of bit-map storage data in bytes. If the bit map is uncompressed, *0* (default) can be specified for this.

cxResolution Horizontal component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution Vertical component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cclrUsed The number of color indexes from the color table that are used by the bit map. If it is *0* (default), all the indexes are used. If it is non-zero, only the first cclrUsed entries in the table are accessed by the system. Further entries can be omitted.

For standard formats with a cBitCount of 1, 4, or 8 (and cPlanes = 1), any indexes beyond cclrUsed are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, cclrUsed is the number of colors used by the bit map.

cclrImportant Minimum number of colors indexes for satisfactory appearance of the bit map. More colors can be used in the bit map, however, it is not necessary to assign them to the device palette. These additional colors can be mapped to the nearest colors available. Zero (default) means that all entries are important. For a 24-bitcount standard format, the cclrImportant colors are also listed in the color table relating to this bit map.

usUnits Units of measure of the horizontal and vertical components of resolution:

BRU_METRIC (Default.) Pels per meter.

usReserved Reserved field. If present, it must be *0*.

usRecording Recording algorithm, the format in which bit-map data is recorded:

BRA_BOTTOMUP (Default.) Scan lines are recorded from bottom-to-top.

usRendering Half-toning algorithm used to record bit-map data that has been digitally half-toned:

BRH_NOTHALFTONED (Default.) Bit-map data not half-toned.

BRH_ERRORDIFFUSION Error diffusion or damped error diffusion algorithm.

BRH_PANDA Processing algorithm for noncoded document acquisition.

BRH_SUPERCIRCLE Super circle algorithm.

cSize1 Size value *1*. If BRH_ERRORDIFFUSION is specified in usRendering, cSize1 is the error damping as a percentage in the range *0 – 100*. A value of 100% indicates no damping. A value of 0% indicates that any errors are not diffused.

If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize1 is the x-dimension of the pattern used in pels.

cSize2 Size value *2*. If BRH_ERRORDIFFUSION is specified in usRendering, this parameter is ignored. If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize2 is the y-dimension of the pattern used in pels.

ulColorEncoding Color encoding:

BCE_RGB (Default.) Each element in the color array is an RGB2 data type.

ulIdentifier Reserved for application use.

argbColor[] Color table array of RGB2 structures:

bBlue
bGreen
bRed
fcOptions Reserved.

Return Codes: On completion, the handling routine must return the bit-map handle (hbm), or GPI_ERROR if an error is detected.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_HDC
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SCAN_START.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: Bit-map size is limited by available memory. The maximum width and height are 64KB. Typically, the following standard bit-map formats are used:

Bitcount	Planes
1	1
4	1
8	1
24	1

All presentation drivers must be able to create and use all of the standard formats. However, presentation drivers for two-color devices will lose the color information.

The DC handle supplied to this function must never be NULL, because bit maps always belong to some device. The bit map is created on the device specified and can be selected to a different device later because the graphics engine can handle transfer of bits from one device to the other. When a presentation driver supports only a single color format, requests for other color bit-map formats are mapped to the supported function. No error is returned.

GreDeviceDeleteBitmap

```
#define INCL_GRE_BITMAPS

BOOL GreDeviceDeleteBitmap (hdc, hbm, pReturns, flOptions, pInstance, lFunction)
```

This function destroys a bit map.

Support: This function must be supported by the presentation driver.

Stack Frame

Note: The handling routine must not use the values passed on the stack in the locations reserved for hdc and pInstance. These locations contain undefined data.

Parameter	Data Type	Description
hdc	Reserved	See Note above.
hbm	ULONG	Handle of bit map to be destroyed.
pReturns	PDELETERETURN	Pointer to returned bit-map parameters.
flOptions	ULONG	Additional information used by the engine when creating or deleting a bit map. See below.
reserved	ULONG	See Note above.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceDeleteBitmap.

pReturns DELETERETURN structure:

pInfo Pointer to a BITMAPINFO or BITMAPINFO2 structure
pBits Pointer to bit map

flOptions The only valid flag is:

CBM_INIT When set, bit-map parameters must be returned in pReturns. This means that before deleting the bit map, the handling routine must translate it into one of the standard formats. The presentation driver must then allocate two blocks of memory, one for the bit map and another for the bit-map parameters and color translation table. The presentation driver can use any of the standard formats. However, it must take into account the parameters originally specified in GreDeviceCreateBitmap. It is recommended that the handling routine use the format that uses the least amount of memory without losing any bit-map information.

When this flag is not set, bit-map data is not returned.

Other flags are reserved and should be ignored by the handling routine.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_HDC
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SCAN_START.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeviceGetAttributes

```
#define INCL_GRE_DEVMISC1

BOOL GreDeviceGetAttributes (hdc, lPrimType, flAttrsMask, pAttrs, pInstance, lFunction)
```

This function queries the attribute values currently set in the device.

Support: This function must be supported by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lPrimType	LONG	Bundle primitive type. See below.
flAttrsMask	ULONG	Attribute mask. See below.
pAttrs	PBUNDLE	Pointer to the fixed-format bundle record to which the attributes are returned. Fields other than those indicated by flAttrsMask must not be modified.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceGetAttributes.

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE	Line attribute bundle, see page 8-3.
PRIM_CHAR	Character attribute bundle, see page 8-6.
PRIM_MARKER	Marker attribute bundle, see page 8-12.
PRIM_AREA	Pattern attribute bundle, see page 8-5.
PRIM_IMAGE	Image attribute bundle, see page 8-11.

flAttrsMask Specifies the attributes to be returned. This mask contains a bit corresponding to each attribute in the bundle record that is required. For each set bit, the handling routine must return the corresponding attribute values and default mask bits. Only the foreground color and background color attributes can be requested for any primitive type.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeviceQueryFontAttributes

```
#define INCL_GRE_DEVMISC2

BOOL GreDeviceQueryFontAttributes (hdc, cMetrics, pfmMetrics, pInstance, lFunction)
```

This function stores the metrics of the currently selected font at the location addressed by pfmMetrics. Notice that the handling routine must transform device coordinates to world coordinates before returning the results to the calling routine. This can be done by using GreConvert.

Support: This function must be supported by the presentation driver. GreDeviceQueryFontAttributes is called from the graphics engine internal function GreQueryFontAttributes in response to an application calling one of the GpiQueryFont*xxx*() APIs. This call can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
cMetrics	ULONG	Size of FONTMETRICS structure
pfmMetrics	PFONTMETRICS	Pointer to FONTMETRICS structure
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceQueryFontAttributes

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeviceQueryFonts

```
#define INCL_GRE_DEVMISC2

LONG GreDeviceQueryFonts (hdc, flOptions, pszFaceName, pfmMetrics, cMetrics, pcFonts, pInstance, lFunction)
```

If the QF_PUBLIC option flag (see below) is set, this function returns the characteristics of device fonts in an array of FONTMETRICS structures. The returned fonts include those that correspond to device modes such as *expanded* and *expanded-bold*. When the DC is not set to draft mode, the returned fonts are those that can be positioned to the nearest pel. Such precision is not necessary if the DC is in *draft mode*. Draft mode is set by the system calling the GreEscape DEVESC_DRAFTMODE.

In the FONTMETRICS structures, the handling routine sets:

- szFacename field to a meaningful name, for example, 'Courier Bold'.
- usCodepage field to *0*. (This field has no significance in this context.)
- lMatch field to a negative value. This allows the presentation driver to map the font when the value is specified in a call to GreRealizeFont.

The presentation driver must transform device coordinates to world coordinates before it returns the results to the calling routine. This can be done by using GreConvert. For presentation drivers that support only outline fonts, the return values are for outline fonts even when image fonts have been loaded.

Support: This function must be supported by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle..
flOptions	ULONG	Option flags. See below.
pszFaceName	PSZ	Pointer to FaceName to match. If this is a NULL pointer, all faces are matched.
pfmMetrics	PFONTMETRICS	Pointer to array of FONTMETRICS structures.
cMetrics	LONG	Number of bytes of each metrics structure in the metrics array.
pcFonts	PLONG	Pointer to the number of fonts requested. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceQueryFonts.

flOptions The only valid flag for this function is:

QF_PUBLIC When this flag is set, the handling routine must return all device fonts. *Device fonts* are public fonts and they are returned in the array addressed by pfmMetrics. If this flag is not set, the handling routine should not return any fonts.

pcFonts This is a pointer to the number of fonts requested. On completion, the handling routine modifies the value indicated to the number of fonts returned. An application can determine the number of public fonts available to it by passing a value of *0* at the address indicated by this pointer.

Return Codes: The handling routine should return the number of fonts not returned, or GPI_ALTERROR if an error occurred.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeviceSelectBitmap

```
#define INCL_GRE_BITMAPS

BOOL GreDeviceSelectBitmap (hdc, hbm, pInstance, lFunction)
```

This function informs the presentation driver that a new bit map is selected into the DC. See also "GreSelectBitmap" on page 11-56.

Support: This function must be supported by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hbm	ULONG	Device bit-map handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceSelectBitmap

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeviceSetAttributes

```
#define INCL_GRE_DEVMISC1

BOOL GreDeviceSetAttributes (hdc, lBType, flDefsMask, flAttrsMask, pAttrs, pInstance, lFunction)
```

This function sets attributes in the attribute bundle specified by lBType. Pointers to the current attribute bundles are maintained by presentation drivers in the instance data structure. The handling routine for GreDeviceSetAttributes modifies the specified bundle as directed by flAttrsMask and flDefsMask (see "Remarks" on page 8-49).

The handling routine must allow any attribute to be set to any value in the defined range for that attribute even when the value cannot be implemented on the device. For example, the presentation driver for a vector hardcopy device must accept BM_XOR background mix. When the hardcopy driver is called to write to the device, it should map values that cannot be implemented to the default value. If this call would set any of the attributes to a value that is not in the defined range of values for that attribute, the handling routine must restore all attributes to the value they had on entry to this routine.

When this function is called for the first time to set the character attributes, the handling routine should set the default font in the usSet parameter of the character attribute bundle (see page 8-6). If the default font is an engine font, the presentation driver must save the address and flags of the font. This ensures that the default font is restored if the DC is reset (see "Enable Subfunction 09H — ResetDCState" on page 7-19).

Support: This function must be supported by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lBType	LONG	Bundle type. See below.
flDefsMask	ULONG	Mask indicating the attributes to be set to their standard default values.
flAttrsMask	ULONG	Mask indicating the attributes to be modified.
pAttrs	PBUNDLE	Pointer to a bundle structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceSetAttributes.

lBType Device attribute bundle. The following device bundles are defined:

PRIM_AREA Pattern attribute bundle, see page 8-5.
PRIM_CHAR Character attribute bundle, see page 8-6.
PRIM_IMAGE Image attribute bundle, see page 8-11.
PRIM_LINE Line attribute bundle, see page 8-3.
PRIM_MARKER Marker attribute bundle, see page 8-12.

All device bundles share a similar format. They consist of two bundles, a bundle of logical attributes and a bundle of device information.

pAttrs Pointer to the DLINEBUNDLE, DCHARBUNDLE, DMARKERBUNDLE, DAREABUNDLE, or DIMAGEBUNDLE structure containing the new attributes.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The parameters flDefsMask and flAttrsMask are instances of the mask for the specified attribute bundle. Valid flags are listed under the bundle definitions on pages 8-3, 8-5, 8-6, 8-11, and 8-12. Flags set in flAttrsMask identify which fields in the attribute bundle are to be changed. For each flag that is set in flAttrsMask, the state of that flag in flDefsMask determines the source for the new value of the field: If the flag is set in both masks, the corresponding field should be set to its default value. If the flag is set in flAttrsMask and not set in flDefsMask, the corresponding field will be set from the relevant field in the bundle addressed by pAttrs.

In the attribute bundle addressed by pAttrs, the only fields that contain valid values are those that will be used to modify the device context's attribute bundle.

When setting pattern and area attributes, the pattern origin from world coordinates must be converted (in the attributes bundle) to device coordinates (in the DC instance data).

GreDeviceSetDCOrigin

```
#define INCL_GRE_DEVMISC3

BOOL GreDeviceSetDCOrigin (hdc, pptlDC, pInstance, lFunction)
```

This function sets the origin of the device context, which when created, has its origin set to *0, 0*.

Support: This function must be supported by presentation drivers for display devices and for hardcopy devices that use banding. The minimum requirement for other hardcopy devices is for the handling routine to return TRUE if the origin addressed by pptlDC is set to *0*, or to log an error and return FALSE.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pptlDC	PPOINTL	Pointer to the DC origin. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceSetDCOrigin.

pptlDC This is the offset to the origin of the device context indicated by hdc. Convert does not add in this offset (see "GreConvert" on page 10-26). Therefore, the presentation driver must add it to all device coordinates to make them screen coordinates.

Note: WORLD_COORDINATE to SCREEN_COORDINATE is not a valid conversion.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeviceSetGlobalAttribute

```
#define INCL_GRE_DEVMISC1

BOOL GreDeviceSetGlobalAttribute (hdc, lAttrType, lAttribute, flOptions, pInstance, lFunction)
```

This function sets the individual primitive attributes to the specified value in the line, area, character, image and marker bundles. If this call sets any attributes to a value that is not in the defined range of values for that attribute, the handling routine must restore all attributes to the value they had on entry to this routine.

Support: This function must be supported by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lAttrType	LONG	Specifies the attribute. See below.
lAttribute	LONG	New attribute value.
flOptions	ULONG	See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceSetGlobalAttribute.

lAttrType Attribute type:

ATYPE_COLOR	Foreground color
ATYPE_BACK_COLOR	Background color
ATYPE_MIX_MODE	Foreground mix
ATYPE_BACK_MIX_MODE	Background mix.

ATYPE_BACK_COLOR and ATYPE_BACK_MIX_MODE do not apply to the line bundle.

flOptions The only allowable option is:

GATTR_DEFAULT When set, the attribute indicated by lAttrType is set to its default value. When this flag is not set, the attribute is set to the value of lAttribute.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_COLOR_ATTR
PMERR_INV_HDC
PMERR_INV_MIX_ATTR.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDisjointLines

```
#define INCL_GRE_LINES

LONG APIENTRY GreDisjointLines (hdc, paptlPoint, cPoints, pInstance, lFunction)
```

This function draws a sequence of disjoint straight lines using the end-point pairs specified. Notice that if COM_TRANSFORM is not set, the pairs are expected in screen coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paptlPoint	PPOINTL	Pointer to an array of cPoints (x, y) pairs containing the end-points for the lines.
cPoints	LONG	Number of (x, y) pairs in the points array. When this is passed as *0*, the handling routine takes no action except to return Successful.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDisjointLines.

Return Codes: On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN.
PMERR_INV_RECT
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDrawBits

```
#define INCL_GRE_BITMAPS

BOOL GreDrawBits (hdc, pBitmap, pInfo, cPoints, paptlPoint, lRop, flOptions, pInstance, lFunction)
```

This function draws a rectangle of bits.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pBitmap	PBYTE	Pointer to bit-map data. See below.
pInfo	PBITMAPINFO	Pointer to BITMAPINFO or BITMAPINFO2 structure defining the new bit map. See below.
cPoints	LONG	Number of (x, y) pairs in paptlPoint; this count must be equal to *4*. See below.
paptlPoint	PPOINTL	Pointer to an array of (x, y) coordinate pairs. See below.
lRop	LONG	Raster operation code. See below.
flOptions	ULONG	Specified treatment of eliminated lines and columns when compression is done. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDrawBits.

pBitmap Pointer to the pel data of the bit map. This data is stored in the order that the coordinates appear on a display screen, that is, the pel in the lower-left corner is the first in the bit map. Pels are scanned to the right, and upward, from that position. The bits of the first pel are stored beginning with the most significant bits of the first byte. The data for pels in each scan line is packed together tightly. However, all scan lines are padded at the end so that each one begins on a ULONG boundary. That is, three bytes of pel data will hold one 24-bit pel, three 8-bit pels, six 4-bit pels, or twenty-four 1-bit pels. If those three bytes are the only pel data for that scan line, one more byte of zeros would be required to pad the line to a ULONG boundary.

pInfo Pointer to either a BITMAPINFO structure:

- **cbFix** Length of structure
- **cx** Bit-map width
- **cy** Bit-map height
- **cPlanes** Number of color planes, *1* if standard format
- **cBitCount** Number of adjacent color bits per pel
- **argbColor[]** Color table array of RGB structures:
  - **bBlue**
  - **bGreen**
  - **bRed.**

Or pointer to a BITMAPINFO2 structure:

- **cbFix** Length of structure

cx Bit-map width

cy Bit-map height

cPlanes Number of color planes, *1* if standard format

cBitCount Number of adjacent color bits per pel

ulCompression Compression scheme used to store the bit map:

BCA_UNCOMP Bit map is uncompressed (the only valid value).

cbImage Length of bit-map storage data in bytes. If the bit map is uncompressed, *0* (default) can be specified.

cxResolution Horizontal component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution Vertical component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cclrUsed The number of color indexes from the color table that are used by the bit map. If it is *0* (default), all the indexes are used. If it is non-zero, only the first cclrUsed entries in the table are accessed by the system; further entries can be omitted.

For standard formats with a cBitCount of 1, 4, or 8 (and cPlanes = 1), any indexes beyond cclrUsed are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, cclrUsed is the number of colors used by the bit map.

cclrImportant Minimum number of colors indexes for satisfactory appearance of the bit map. More colors can be used in the bit map but it is not necessary to assign them to the device palette. These additional colors can be mapped to the nearest colors available. Zero (default) means that all entries are important. For a 24-bitcount standard format, the cclrImportant colors are also listed in the color table relating to this bit map.

usUnits Units of measure of the horizontal and vertical components of resolution:

BRU_METRIC (Default.) Pels per meter.

usReserved Reserved field. If present, it must be *0*.

usRecording Recording algorithm, the format in which bit-map data is recorded:

BRA_BOTTOMUP (Default.) Scan lines are recorded from bottom-to-top.

usRendering Half-toning algorithm used to record bit-map data that has been digitally half-toned:

BRH_NOTHALFTONED (Default.) Bit-map data not half-toned.

BRH_ERRORDIFFUSION Error diffusion or damped error diffusion algorithm

BRH_PANDA Processing algorithm for noncoded document acquisition

BRH_SUPERCIRCLE Super circle algorithm

cSize1 Size value *1*. If BRH_ERRORDIFFUSION is specified in usRendering, cSize1 is the error damping as a percentage in the range *0—100*. A value of 100% indicates no damping. A value of 0% indicates that any errors are not diffused.

If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize1 is the x-dimension of the pattern used in pels.

cSize2 Size value *2*. If BRH_ERRORDIFFUSION is specified in usRendering, this parameter is ignored. If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize2 is the y-dimension of the pattern used in pels.

ulColorEncoding Color encoding:

BCE_RGB (Default.) Each element in the color array is an RGB2 data type.

ulIdentifier Reserved for application use.

argbColor[] Color table array of RGB2 structures:

bBlue
bGreen
bRed
fcOptions Reserved. Must be *0*.

cPoints = 4 The operation is determined by comparing the sizes of the two rectangles:

Target<Source Compress the source rectangle into the target rectangle. The flOptions flags determine how eliminated rows and columns should be handled.

Target = Source Copy between equal rectangles.

Target>Source Stretch the source rectangle into the target rectangle. In this case the call can be passed to the GreBitblt routine in the graphics engine.

paptlPoint A pointer to a block of (x, y) coordinate pairs that define the target and source rectangles. The coordinates, which can be passed as a pair of RECTL structures, define the bottom-left and top-right corners of the target and source rectangles (see cPoints above for more information):

```
(xTgtBL, yTgtBL), (xTgtTR, yTgtTR), (xSrcBL, ySrcBL), (xSrcTR, ySrcTR)
```

When the source rectangle is totally or partially outside the source bit map (or device), the operation is implementation-dependent for that area, that is, the user must decide what to do.

Note: When BBO_TARGWORLD is set, the target rectangle is inclusive at all boundaries. The source is non-inclusive.

When BBO_TARGWORLD is not set, the rectangles are non-inclusive. That is, they include the left and lower boundaries in device units but not the top and right boundaries. When the bottom-left corner of a rectangle maps to the same device pel as the top-right corner, that rectangle is considered to be empty.

lRop Raster operation code. The low-order byte represents a mix value in the range 00H—FFH. Raster operation code values and the mix-bit table are defined in the *OS/2 2.0 Presentation Manager Programming Reference*. The handling routine uses lRop to determine the operations to perform on pattern, source and target to get the required mix.

In addition to the ROP values defined at the API, the presentation driver must support ROP_GRAY (000080CAH). This value is used to shade the text for menu items that are not

currently selectable. When ROP_GRAY is set, the handling routine overpaints the foreground pattern using the current pattern and the background pattern color (background pels for the pattern are not changed). For the PATSYM_HALFTONE pattern, this overpaints the background pattern color onto alternate pels leaving those in between unchanged.

flOptions Option flags:

BBO_OR	Stretch and compress, as necessary, ORing any eliminated rows and columns. Used for White on Black.
BBO_AND	Stretch and compress, as necessary, ANDing any eliminated rows and columns. Used for Black on White.
BBO_IGNORE	Stretch and compress, as necessary, ignoring any eliminated rows and columns. Used for color.
BBO_TARGWORLD	The target rectangle is defined in world coordinates in the target PS. When this option is specified, the target rectangle is transformed to device coordinates. Where any shear or rotation has occurred, the target rectangle must be converted to an upright rectangle that bounds the transformed figure. This is then used as the target for the operation. No inversion of the image takes place.
BLTMODE_SRC_BITMAP	hdcSrc is a bit-map handle. The bit map must not be currently selected into a device context. If this flag is not set, hdcSrc is a DC handle.
BLTMODE_ATTRS_PRES	If set, the pBattrs parameter is present. This option can be ORed with any of the options above.

Note: Flags 15–31 are not used by the system; they are reserved for use by the presentation driver.

Return Codes: On completion, the handling routine must return a LONG integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful
GPI_HITS	Successful with correlate hit (returned by the display driver when the correlate flag is *on*, and a hit is detected)
GPI_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_HDC_BUSY
PMERR_INCORRECT_DC_TYPE
PMERR_INV_BITBLT_MIX
PMERR_INV_BITBLT_STYLE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDrawBorder

```
#define INCL_GRE_BITMAPS

BOOL GreDrawBorder (hdc, prclFrame, cxBorder, cyBorder, clrBorder, clrInterior, flCmd, pInstance, lFunction)
```

This fast-frame function draws an internal border in a rectangular frame. The interior can also be filled.

Support: This function must be hooked by all presentation drivers. Drivers for hardcopy devices do nothing except return TRUE (Successful). Display drivers must return Failure if fast-frame drawing is not supported. GreDrawBorder is not called by any specific function. It is used to do fast-frame drawing.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclFrame	PRECTL	Pointer to RECTL structure defining the frame in screen coordinates.
cxBorder	ULONG	Thickness of vertical border in device coordinates.
cyBorder	ULONG	Thickness of horizontal border in device coordinates.
clrBorder	LONG	Color of border in any valid format.
clrInterior	LONG	Color of interior in any valid format.
flCmd	ULONG	Options. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDrawBorder.

prclFrame Pointer to a RECTL structure. Coordinates are passed in screen coordinates, which are inclusive at the bottom-left corner and exclusive at the top-right corner.

flCmd Valid values are:

DB_ROP An option flag that defines the mix to be used for the border and the interior. The mixes are mutually exclusive with one another but can be combined independently with DB_INTERIOR or DB_AREAATTRS. This option can have any of the following values:

- **DB_PATCOPY** (Default.) Use ROP_PATCOPY (see "GreBitblt" on page 8-26). This is a copy of the pattern to the destination.
- **DB_PATINVERT** Exclusive-OR the current pattern and the destination (ROP_PATINVERT). Current mix and color parameters are ignored.
- **DB_DESTINVERT** Inverts the destination (ROP_DESTINVERT).
- **DB_AREAMIXMODE** Maps the current area foreground-mix attribute to a Bitblt raster operation. The area background-mix mode is ignored.

DB_INTERIOR Fills the area defined by prclFrame excluding the border defined by cxBorder and cyBorder.

DB_AREAATTRS When set, the pattern used for the border is the one currently defined in the area attribute. The pattern used for the interior is the one that would be obtained by calling GreSetAttributes with the area attribute background color passed for the foreground color, and the area attribute foreground color passed for the background. See "GreSetAttributes" on page 11-58.

When this flag is not set (default), the border pattern is equivalent to using GreSetAttributes for the area attributes, which use clrBorder as foreground color and clrInterior as the background. The Interior pattern is equivalent to using GreSetAttributes for the area attributes, which use clrInterior color as foreground color and clrBorder as the background.

The handling routine should ignore the remaining flags, DB_STANDARD and DB_DLGBORDER, which are used by the Frame Manager.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INCOMPATIBLE_BITMAP
PMERR_INCORRECT_DC_TYPE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BITBLT_MIX
PMERR_INV_BITBLT_STYLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_INDEX
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_DC_DATA
PMERR_INV_DC_TYPE
PMERR_INV_DRAW_BORDER_OPTION
PMERR_INV_DRIVER_NAME
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT

PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_SCAN_START
PMERR_INV_USAGE_PARM
PMERR_UNSUPPORTED_ATTR_VALUE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The parameters, cxBorder and cyBorder, can be passed as *0*. When both are *0*, the interior must still be drawn. When the x-borders and y-borders overlap, the border is drawn as a single rectangle with no interior.

GreDrawBorder is a Bitblt accelerator and is similar in function and limitation to GreBitblt, GreDeviceSetAttributes, and GreSetAttributes.

See the *OS/2 2.0 Presentation Manager Programming Reference* for a description of the WinDrawBorder function.

GreDrawLinesInPath

```
#define INCL_GRE_LINES

LONG GreDrawLinesInPath (hdc, prclBound, pLine, cLines, pInstance, lFunction)
```

This function draws a sequence of one or more straight lines from the sequence of linked structures addressed by pLine. The structures can be a mixture of LINE, CURVE, and FILLETSHARP structures. These have similar forms and the second field, bType, identifies the type of the structure. Starting at the structure addressed by pLine, the handling routine examines the bType field of each structure in turn. If bType is LINE_IDENTIFIER, the handling routine draws the line. Otherwise, it uses the value of the npcvNext field to skip to the next structure. This process continues until the handling routine has drawn the number of lines specified by cLines. Before drawing a line, the handling routine must check the CURVE_DO_FIRST_PEL flag to determine whether it should draw the first pel of the line. When the line passes between two pel positions, the presentation driver should round down to the nearest pel for values of *0.5* or less.

The call to GreDrawLinesInPath in the presentation driver is made by the graphics engine. Notice that the coordinates are passed as screen coordinates (device coordinates + DC origin) and the lines are already completely clipped.

Support: This function must be supported by all presentation drivers. However, it is not called by any specific function. GreDrawLinesInPath is used by the filing routines and the path rendering routines within the graphics engine to produce output.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclBound	PRECTL	Pointer to a bounding box for the lines.
pLine	PCURVE	Pointer to the first of a series of linked structures. See below.
cLines	ULONG	Count of LINE structures to draw.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDrawLinesInPath.

prclBound Pointer to a RECTL structure:

- **xLeft** Minimum x-coordinate of rectangle
- **yBottom** Minimum y-coordinate of rectangle
- **xRight** Maximum x-coordinate of rectangle
- **yTop** Maximum y-coordinate of rectangle

pLine Pointer to the first structure in the path definition containing the lines that have to be drawn. The LINE structure, shown below, is an example of the more general CURVE structure. These two structures and the FILLETSHARP structure are defined in header file. They all take the same general form and are distinguished by the value of bType.

The LINE structure is defined as:

- **lIdent** Identifier. This value has no significance for the presentation driver.
- **bType** Structure type. The only significant value is LINE_IDENTIFIER, which indicates that this structure is a LINE structure. If any other value is detected, the handling routine should skip to the structure addressed by npcvNext.

ulStyle	Line style. See "Line Attributes" on page 8-3.
fl	Flags. The only significant flag is:
	CURVE_DO_FIRST_PEL When set, the handling routine must draw the first pel in the line. See also "First and Last Pel Considerations" on page 8-21.
pcvNext	Pointer to next structure in the sequence.
pcvPrev	Pointer to previous structure.
Reserved1[2]	Reserved parameter.
ptlfxA	Start of the already clipped line (inclusive).
ptlfxC	End of the already clipped line (inclusive).
ptlA	Start point of unclipped line (inclusive).
ptlC	End point of unclipped line (inclusive).
lRslope	Ignored by the presentation driver.
Reserved2[4]	Reserved parameter.

Return Codes: On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreErasePS

```
#define INCL_GRE_DEVMISC2

BOOL GreErasePS (hdc, pInstance, lFunction)
```

This function resets the presentation space of the device context to the background color CLR_BACKGROUND. Hardcopy drivers should return TRUE without taking any action. The handling routine does not update GPI_BOUNDS or return correlation data, and it is not affected by the PCTL_DRAW control or COM_DRAW command flag. However, in display drivers, the handling routine should update USER_BOUNDS if the COM_ALT_BOUND command flag is set.

Support: This function must be supported by the presentation driver. GreErasePS is called from GpiErase, and is used to erase the contents of the presentation space currently associated with the device context. Hardcopy drivers should not take any action except to return TRUE (Successful).

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreErasePS

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function is subject to all clipping.

GreEscape DEVESC_ABORTDOC (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_ABORTDOC, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function aborts the current document. The handling routine in the presentation driver discards all data (such as data in a spooler buffer or journal file) received for the current document and closes any files associated with it. The *current document* is defined as any data back to, and including, the DEVESC_STARTDOC statement.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_ABORTDOC
cInCount	LONG	The handling routine ignores this parameter
pInData	PBYTE	The handling routine ignores this parameter
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: The following are three possible abort scenarios and the steps to be taken by a hardcopy driver:

1. The user aborts a job:
   a. The spooler calls SplQpControl(SPLC_ABORT).
   b. The queue driver calls DevEscape(DEVESC_ABORTDOC).
   c. The hardcopy driver sets a flag indicating that the job was aborted and returns the DevEscape(DEVESC_ABORTDOC) thread. The next time PrtWrite returns, the hardcopy driver completes the current page, for example, by sending a form feed and some null data to make sure the printer is not in graphics mode.
   d. The hardcopy driver calls PrtClose.

   Note: The hardcopy driver must be able to accept DEVESC_ABORT while processing data or DEVESC_ENDDOC.
2. The printer runs out of paper or is offline:
   a. The spooler function, PrtWrite, fails and returns an error to the hardcopy driver.
   b. If the job was aborted, the hardcopy driver calls PrtClose. Otherwise, it calls SplMessageBox.
   c. The spooler brings up a message box or sends a message to the user, holds the job, and waits on a semaphore until the job is released (possibly across the network), in which case it will return

RETRY. If the user selects ABORT, the hardcopy driver calls PrtAbort and PrtClose. If the user selects RETRY, the hardcopy driver will try PrtWrite again (see above).

Note: The spooler will ignore all PrtWrite operations after PrtAbort is called. After the hardcopy driver has called PrtClose, it should return errors until the DC is closed.

3. The application aborts the job while spooling:

   a. The application calls DevEscape(DEVESC_ABORTDOC).
   b. The hardcopy driver calls SplQmAbortDoc.
   c. The hardcopy driver flags the DC as being aborted.
   d. The application calls either DevCloseDC or DevEscape(DEVESC_STARTDOC) to start another job on the same DC.

Note: This escape code is metafiled but not recorded.

GreEscape DEVESC_BREAK_EXTRA (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_BREAK_EXTRA, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

Note: This escape code is implemented by old hardcopy drivers. The function of this escape code is replaced by the character attribute, fxBreakExtra. See "Character Attributes" on page 8-6.

This function changes the width of the break character on a hardcopy device. The handling routine sets or resets, as determined by the value of cInCount, an extra width value for the break character. Upon completion, the width of the break character is the default width specified by the font plus any extra widths set by DEVESC_BREAK_EXTRA and DEVESC_CHAR_EXTRA. The extra widths can be positive, zero, or negative.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lEscape	LONG	DEVESC_BREAK_EXTRA.
cInCount	LONG	Number of bytes pointed to by pInData. When cInCount=0, the handling routine resets the extra width to *0*.
pInData	PBYTE	If cInCount is not equal to *0*, pInData is a pointer to a FIXED value. This value is the required extra width defined in world coordinates.
pcOutCount	PLONG	The handling routine ignores this parameter.
pOutData	PLONG	The handling routine ignores this parameter.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD=flags; low-order WORD=NGreEscape.

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled and recorded.

GreEscape DEVESC_CHAR_EXTRA (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_CHAR_EXTRA, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

Note: This escape code is implemented by old hardcopy drivers. The function of this escape code is replaced by the character attribute, fxExtra. See "Character Attributes" on page 8-6.

This function changes the width of all characters including the break character on a hardcopy device. The handling routine sets or resets, as defined by the value of cInCount, an extra width value. Upon completion, the width of a character is the default width specified by the font plus the extra width set by DEVESC_CHAR_EXTRA. The extra width can be positive, zero, or negative.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lEscape	LONG	DEVESC_CHAR_EXTRA.
cInCount	LONG	Number of bytes pointed to by pInData. When cInCount=0, the handling routine should reset the extra width to *0*.
pInData	PBYTE	If cInCount is not equal to *0*, pInData is a pointer to a FIXED value. This value is the required extra width defined in world coordinates.
pcOutCount	PLONG	The handling routine ignores this parameter.
pOutData	PLONG	The handling routine ignores this parameter.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD=flags; low-order WORD=NGreEscape.

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled and recorded.

GreEscape DEVESC_DBE_FIRST (DBCS Support)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_DBE_FIRST, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function informs the presentation driver that character codes for subsequent output data will use two bytes per character.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_DBE_FIRST
cInCount	LONG	The handling routine ignores this parameter
pInData	PBYTE	The handling routine ignores this parameter
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: For an OD_METAFILE device, DEVESC_DBE_FIRST is passed to the presentation driver but is not metafiled. For an OD_QUEUED device with PM_Q_STD, the spooler records this escape and it is not passed to the presentation driver.

GreEscape DEVESC_DBE_LAST (DBCS Support)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_DBE_LAST, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function informs the presentation driver that character codes for subsequent output data will use one byte per character.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_DBE_LAST
cInCount	LONG	The handling routine ignores this parameter
pInData	PBYTE	The handling routine ignores this parameter
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: For an OD_METAFILE device, DEVESC_DBE_LAST is passed to the presentation driver but is not metafiled. For an OD_QUEUED device with PM_Q_STD, the spooler records this escape and it is not passed to the presentation driver.

GreEscape DEVESC_DRAFTMODE (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_DRAFTMODE, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

Note: This escape code is implemented by old hardcopy drivers and is being phased out.

This function sets draft mode (text mode) *on* or *off*. Setting draft mode *on*, tells the presentation driver that the coming page need not contain any graphics or all-points-addressable output. This escape code is valid only at a page boundary, for example, after DEVESC_STARTDOC or DEVESC_NEWFRAME.

When draft mode is *on*, the presentation driver can choose to optimize throughput by:

- Ignoring all graphics primitives such as lines, arcs, and areas
- Using the fonts provided by the output device
- Approximating positions received in calls to functions such as GreSetCurrentPosition and GreCharStringPos to the nearest character position that the output device supports for the current font.

The presentation driver must maintain current attributes such as color, mix, and transforms when draft mode is *on* even though they might have no effect on the draft output. Similarly, the driver needs to track font changes and respond by setting the appropriate device font such as enlarged, condensed, or italic.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler records this escape in the buffer and does not pass it to the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lEscape	LONG	DEVESC_DRAFTMODE.
cInCount	LONG	Number of bytes pointed to by pInData. This is *2*.
pInData	PBYTE	Short integer value specifying the mode, *1* for draft mode *on*, *0* for *off*.
pcOutCount	PLONG	The handling routine ignores this parameter.
pOutData	PLONG	The handling routine ignores this parameter.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape.

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled and recorded.

GreEscape DEVESC_ENDDOC (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_ENDDOC, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function ends the current document. The handling routine does whatever work is required to complete the job. For example, a DC for an OD_QUEUED device with PM_Q_STD data would close the spooler buffer and transfer the buffered data to a spool file. As with DEVESC_STARTDOC, do not assume that this escape code is always issued at the end of a document. When it has not been issued, the DEVESC_ENDDOC work must be done in the BeginCloseDC or CloseDC routine.

Note: DEVESC_ENDDOC is mandatory at the API when writing PM_Q_STD data to an OD_QUEUED device.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lEscape	LONG	DEVESC_ENDDOC.
cInCount	LONG	The handling routine ignores this parameter.
pInData	PBYTE	The handling routine ignores this parameter.
pcOutCount	PLONG	Pointer. On input, this specifies the size of the buffer pointed to by pOutData. On output, it is set to the number of data bytes returned in this buffer. The input value of this parameter is usually 2. On completion, this is set to *0* if no ID is returned in pOutData.
pOutData	PLONG	Pointer to a data area in which the Job ID of the spooled print job is returned. Set to NULL if there is no Job ID, for example, when the hardcopy DC is OD_DIRECT.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape.

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled but not recorded.

GreEscape DEVESC_FLUSHOUTPUT (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_FLUSHOUTPUT, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

Note: This escape code is implemented by old hardcopy drivers and is being phased out.

This function flushes any output received for the current document. The handling routine discards all data (such as data stored in a spooler buffer or journal file) received for the current document.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_FLUSHOUTPUT
cInCount	LONG	The handling routine ignores this parameter
pInData	PBYTE	The handling routine ignores this parameter
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled and recorded.

GreEscape DEVESC_GETCP (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_GETCP, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

Note: This escape code is used only by hardcopy drivers with built-in fonts.

This function gets a hardcopy data stream that would set the specified code page in the hardcopy device. The data stream should cater for options such as *draft* and *NLQ* as defined in the OS2_PM_DRV_DEVMODE dialog. The handling routine in the presentation driver responds by writing the data stream in the buffer addressed by pOutData and the count of bytes into the LONG value addressed by pcOutCount.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lEscape	LONG	DEVESC_GETCP.
cInCount	LONG	The handling routine ignores this parameter.
pInData	PBYTE	The handling routine ignores this parameter.
pcOutCount	PLONG	Pointer to a value that shows the number of bytes addressed by pOutData. The handling routine should update this value to show the number of bytes returned in the buffer.
pOutData	PLONG	Pointer to the buffer in which the handling routine returns the required data.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape.

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is not metafiled or recorded.

GreEscape DEVESC_GETSCALINGFACTOR (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_GETSCALINGFACTOR, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function gets the incremental scaling factors for the (x, y) axes of a hardcopy device. The handling routine returns the scaling factors as an SFACTORS structure at the location addressed by pOutData and changes the value addressed by pcOutCount to show the number of bytes in the returned structure.

Scaling factors are used for physical devices where one unit on the x axis is not equal to one unit on the y axis. The factors show an arbitrary unit length expressed in x units and y units. The length is chosen so that the number of x units and y units can be expressed as an exponent of 2 and the exponents are returned in the SFACTORS structure. For example, if there are 8 units of *x* in the arbitrary unit length, lXscale is set to *3*.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler passes this escape to the presentation driver without recording it.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lEscape	LONG	DEVESC_GETSCALINGFACTOR.
cInCount	LONG	The handling routine ignores this parameter.
pInData	PBYTE	The handling routine ignores this parameter.
pcOutCount	PLONG	Number of bytes pointed to by pOutData. On return, this is updated by the handling routine to the number of bytes actually returned.
pOutData	PLONG	Pointer to SFACTORS structure: **lXscale** X-scaling factor, as an exponent of 2. **lYscale** Y-scaling factor, as an exponent of 2.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape.

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is not metafiled or recorded.

GreEscape DEVESC_NEWFRAME (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_NEWFRAME, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function indicates that there is no more data for the current page. For a display device, DEVESC_NEWFRAME is similar to GreErasePS. However, the handling routine should reset the attributes (color and mix). For hardcopy devices, the handling routine would advance the paper to a new page.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler records this escape in the buffer and does not pass it on to the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_NEWFRAME
cInCount	LONG	The handling routine ignores this parameter
pInData	PBYTE	The handling routine ignores this parameter
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: The following sequence implies that the previous page is completed and ejected, and then a blank page is ejected, even if no other functions are called between the two GreEscapes.

```
GreEscape(DEVESC_NEWFRAME)
GreEscape(DEVESC_NEWFRAME)
```

This escape code is metafiled and recorded.

GreEscape DEVESC_NEXTBAND (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_NEXTBAND, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

Note: This escape code is implemented by old hardcopy drivers that support banding and is being phased out.

This function indicates that there is no more data for the current band and gets the coordinates of the next band. If there is no current band, the handling routine returns the coordinates of the first band. DEVESC_NEXTBAND is used by programs that do their own banding. It is not necessary for hardcopy drivers for devices, which cannot use banded output, to support this escape code. See "Banding" on page 2-7. Notice that DEVESC_NEWFRAME is issued to start a new page.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_NEXTBAND
cInCount	LONG	The handling routine ignores this parameter
pInData	PBYTE	The handling routine ignores this parameter
pcOutCount	PLONG	Number of bytes of data pointed to by pOutData. On return, this is updated to the number of bytes actually returned.
pOutData	PLONG	Pointer to a BANDRECT structure where the device coordinates of the next band are returned: **xLeft** X-coordinate of the lower-left corner of the rectangular band **yBottom** Y-coordinate of the lower-left corner of the rectangular band **xRight** X-coordinate of the top-right corner of the rectangular band **yTop** Y-coordinate of the top-right corner of the rectangular band. An empty rectangle (xLeft=xRight or yTop=yBottom) marks the end of the banding operation).
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD=flags; low-order WORD=NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled but not recorded.

GreEscape DEVESC_QUERYESCSUPPORT

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_QUERYESCSUPPORT, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function queries whether a particular escape code is implemented by the presentation driver. The presentation driver returns DEV_OK if the specified escape is supported.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler passes this escape on to the presentation driver without recording it.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_QUERYESCSUPPORT
cInCount	LONG	Number of bytes pointed to by pInData
pInData	PBYTE	Pointer to an escape code value specifying the escape function to be checked
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is not metafiled or recorded.

GreEscape DEVESC_QUERYVIOCELLSIZES (Display Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_QUERYVIOCELLSIZES, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function obtains the details of the VIO cell sizes supported by the presentation driver. The DEVESC_QUERYVIOCELLSIZES must be implemented by presentation drivers for display devices that provide two or more VIO fonts with different cell sizes. Other presentation drivers should not implement this escape code.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lEscape	LONG	DEVESC_QUERYVIOCELLSIZES.
cInCount	LONG	The handling routine ignores this parameter.
pInData	PBYTE	The handling routine ignores this parameter.
pcOutCount	PLONG	Pointer. See below.
pOutData	PLONG	Pointer. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape.

pcOutCount Pointer to the number of bytes pointed to by pOutData. On return, the handling routine updates the value indicated by this pointer to the number of bytes actually returned. This value must be an even multiple of sizeof(LONG). When the value passed is less than sizeof(LONG), the handling routine must change it to *0*. Nothing is loaded at the address indicated by pOutData.

When the value passed equals sizeof(LONG), the handling routine must return the number of supported VIO character-cell sizes. The value indicated by pcOutCount is unchanged. The contents of the address indicated by pOutData are updated so that maxcount is the number of VIO cell sizes provided by the device. When the value passed is greater than sizeof(LONG), the handling routine must update the buffer addressed by pOutData so that:

- *maxcount* is the number of VIO cell sizes provided by the device.
- *count* is the number of VIOFONTCELLSIZE structures returned. This can be *0* when OutCount is 2*sizeof(LONG). For example:

```
count==((pcOutCount-2*sizeof(LONG))/2*sizeof(LONG))
```

pOutData Pointer to the address of the data returned. The handling routine stores at this location the following structures:

maxcount Total number of VIO cell sizes provided by the device
count Number of VIOFONTCELLSIZE structures that follow.

Followed by an array of VIOFONTCELLSIZE structures:

xWidth Width of the VIO character cell
yHeight Height of the VIO character cell.

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is not metafiled or recorded.

GreEscape DEVESC_RAWDATA (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_RAWDATA, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function sends device-specific data direct to the spooler or device. The action taken by the handling routine is determined by the DC type. For example, OD_DIRECT DC would send the raw data directly to the device using the Prt*xxx* APIs. See "File System Emulation" on page 4-9.

As a general rule, an application should use DEVESC_RAWDATA only for a complete document or frame within a document. DEVESC_RAWDATA must not be mixed with other drawing functions. If DEVESC_RAWDATA and other drawing functions are called in a single frame, the results are dependent on the implementation. For example, the presentation driver might choose to print the raw data and ignore the other drawing calls.

Note: For an OD_QUEUED device with PM_Q_STD data, the spooler records this call in the buffer and does not pass it on to the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_RAWDATA
cInCount	LONG	Number of bytes pointed to by pInData
pInData	PBYTE	Pointer to raw data
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled and recorded.

GreEscape DEVESC_SETMODE (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_SETMODE, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function sets the printer into a particular mode. It is optional for hardcopy drivers to support this GreEscape. However, those hardcopy drivers that do support it need to know the code page of any of the built-in fonts. For example, if only Code Page 437 is built in, it is the code page used if 437 is requested by GreEscape DEVESC_SETMODE. If Code Page 865 is requested, a suitable code page or font could be downloaded.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_SETMODE
cInCount	LONG	Number of bytes pointed to by pInData
pInData	PBYTE	Pointer to the buffer that contains an ESCSETMODE structure
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled and recorded.

GreEscape DEVESC_STARTDOC (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_STARTDOC, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

This function starts a new document. The handling routine in the presentation driver does whatever initialization is required to spool or print the document. The driver continues to spool or print data until it receives DEVESC_ENDDOC. This ensures that documents longer than one page are not interspersed with other jobs.

Because DEVESC_STARTDOC is not mandatory at the API, it cannot be assumed that an application (when opening a DC for printing) will pass DEVESC_STARTDOC to the presentation driver. In this case, the presentation driver must initialize a spool file with no name or a journal file in the CompleteOpenDC handling routine. If this is done, the presentation driver should set a flag so that initialization is not repeated if DEVESC_STARTDOC is received. Notice that a handling routine is required for DEVESC_STARTDOC to save the specified document name in the DC instance data.

Note: DEVESC_STARTDOC is mandatory at the API for an OD_QUEUED device with PM_Q_STD data.

When this function call is issued for an OD_QUEUED device, the presentation driver must start the recording of data into the spool file by calling SplStdStart. It should also call SplQmStartDoc to pass the name of the spool file to the visual spooler. See "SplStdStart" on page 4-34 and "SplQmStartDoc" on page 4-26.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_STARTDOC
cInCount	LONG	Number of bytes pointed to by pInData
pInData	PBYTE	Pointer to a string containing the name of the document
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

Remarks: This escape code is metafiled but not recorded.

GreEscape DEVESC_STD_JOURNAL (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreEscape (hdc, DEVESC_STD_JOURNAL, cInCount, pInData, pcOutCount, pOutData, pInstance, lFunction)
```

Note: This escape is implemented by old hardcopy drivers and is being phased out.

This function sends a standard journal file to the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lEscape	LONG	DEVESC_STD_JOURNAL
cInCount	LONG	Number of bytes pointed to by pInData
pInData	PBYTE	Pointer to journal data
pcOutCount	PLONG	The handling routine ignores this parameter
pOutData	PLONG	The handling routine ignores this parameter
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEscape

Return Codes: The handling routine returns:

DEV_OK	Successful
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEVESC_ERROR	Error.

GreGetBitmapBits

```
#define INCL_GRE_BITMAPS

LONG GreGetBitmapBits (hdc, hbm, lScanStart, cScanCount, pBitmap, pInfo, pInstance, lFunction)
```

This function can be called for two reasons. If the pointer to application storage (pBitmap) is not NULL, the function transfers bit-map data from a bit map to application storage. If pBitmap is NULL, this function must only fill in the RGB values in the BITMAPINFO or BITMAPINFO2 data structure, which is pointed to by pInfo, and then return the value *0*.

The bit map can be specified by a bit-map handle, or (if this is NULL) a DC handle, in which case the device context must be a memory DC with a bit map currently selected.

Support: This function must be supported by the presentation driver. GreGetBitmapBits is called from GpiQueryBitmapBits, and is used to transfer bit-map data from the device context to application storage. It can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hbm	HBITMAP	Bit-map handle. If *0*, the DC bit map is used.
lScanStart	LONG	Scan line number from where data transfer starts, *0* is the first.
cScanCount	LONG	Number of scan lines to be transferred.
pBitmap	PBYTE	Pointer to bit-map data or NULL. See below.
pInfo	PBITMAPINFO	Pointer to BITMAPINFO or BITMAPINFO2 structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetBitmapBits.

pBitmap Pointer to the pel data of the bit map. This data is stored in the order that the coordinates appear on a display screen, that is, the pel in the lower-left corner is the first in the bit map. Pels are scanned to the right, and upward, from that position. The bits of the first pel are stored beginning with the most significant bits of the first byte. The data for pels in each scan line is packed together tightly. However, all scan lines are padded at the end so that each one begins on a ULONG boundary. That is, three bytes of pel data will hold one 24-bit pel, three 8-bit pels, six 4-bit pels, or twenty-four 1-bit pels. If those three bytes are the only pel data for that scan line, one more byte of *0*s would be required to pad the line to a ULONG boundary.

pInfo Pointer to either a BITMAPINFO structure:

- **cbFix** Length of structure
- **cx** Bit-map width. This value is updated by the handling routine.
- **cy** Bit-map height. This value is updated by the handling routine.
- **cPlanes** Number of color planes, *1* if standard format
- **cBitCount** Number of adjacent color bits per pel
- **argbColor[]** Color table array of RGB structures:
  - **bBlue**
  - **bGreen**
  - **bRed.**

Or pointer to a BITMAPINFO2 structure:

cbFix Length of structure

cx Bit-map width

cy Bit-map height

cPlanes Number of color planes, *1* if standard format

cBitCount Number of adjacent color bits per pel

ulCompression Compression scheme used to store the bit map:

BCA_UNCOMP Bit map is uncompressed (the only valid value).

cbImage Length of bit-map storage data in bytes. If the bit map is uncompressed, *0* (default) can be specified for this.

cxResolution Horizontal component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution Vertical component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group, the bit map that best matches the characteristics of the current output device.

cclrUsed The number of color indexes from the color table that are used by the bit map. If it is *0* (default), all the indexes are used. If it is non-zero, only the first cclrUsed entries in the table are accessed by the system; further entries can be omitted.

For standard formats with a cBitCount of 1, 4, or 8 (and cPlanes = 1), any indexes beyond cclrUsed are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, cclrUsed is the number of colors used by the bit map.

cclrImportant Minimum number of colors indexes for satisfactory appearance of the bit map. More colors can be used in the bit map but it is not necessary to assign them to the device palette. These additional colors can be mapped to the nearest colors available. Zero (default) means that all entries are important. For a 24-bitcount standard format, the cclrImportant colors are also listed in the color table relating to this bit map.

usUnits Units of measure of the horizontal and vertical components of resolution:

BRU_METRIC (Default.) Pels per meter.

usReserved Reserved field. If present, it must be *0*.

usRecording Recording algorithm, the format in which bit-map data is recorded:

BRA_BOTTOMUP (Default.) Scan lines are recorded from bottom-to-top.

usRendering Half-toning algorithm used to record bit-map data that has been digitally half-toned:

BRH_NOTHALFTONED (Default.) Bit-map data not half-toned.

BRH_ERRORDIFFUSION Error diffusion or damped error diffusion algorithm

BRH_PANDA Processing algorithm for noncoded document acquisition

BRH_SUPERCIRCLE Super circle algorithm

cSize1 Size value *1*. If BRH_ERRORDIFFUSION is specified in usRendering, cSize1 is the error damping as a percentage in the range *0—100*. A value of 100% indicates no damping. A value of 0% indicates that any errors are not diffused.

If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize1 is the x-dimension of the pattern used in pels.

cSize2 Size value 2. If BRH_ERRORDIFFUSION is specified in usRendering, this parameter is ignored. If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize2 is the y-dimension of the pattern used in pels.

ulColorEncoding Color encoding:

BCE_RGB (Default.) Each element in the color array is an RGB2 data type.

ulIdentifier Reserved for application use.

argbColor[] Color table array of RGB2 structures:

bBlue
bGreen
bRed
fcOptions Reserved. Must be *0*.

Return Codes: On completion, the handling routine must return a LONG value (lLines), indicating the number of lines transferred, or GPI_ALTERROR if an error occurred.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INCORRECT_DC_TYPE
PMERR_INV_HBITMAP
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SCAN_START.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When the bit-map handle is NULL, the DC must be a memory DC with a bit map currently selected. Otherwise, the DC handle must be valid for that device. The BITMAPINFO or BITMAPINFO2 structure must be initialized with the values of cPlanes and cBitcount for the format of data required. This must be one of the standard formats or a device-specific format that matches the DC. On return, cx, cy, and argbColors are supplied by the system. Conversion of the bit-map data is carried out, if necessary.

pBitmap must point to a storage area large enough to contain data for the requested number of scan lines. The amount of storage required for one scan line can be determined by calling GetBitmapParameters:

```
((cBitcount * cx + 31)/32) * cPlanes * 4 bytes
```

GreGetBoundsData

```
#define INCL_GRE_DEVMISC3

BOOL GreGetBoundsData (hdc, flOptions, pBoundsData, pInstance, lFunction)
```

This function stores the bounding rectangle of previous drawing primitives at the address indicated by pBoundsData. All presentation drivers must support GPI bounds. These bounds should be transformed to model space coordinates when they are accumulated. Display drivers must also support user bounds in screen coordinates. Bounds are inclusive. A NULL boundary is represented by the minimum coordinates of the rectangle, which are greater than its maximum coordinates. If the bounds have been reset, a NULL value is returned for pBoundsData.

Support: This function must be supported by the presentation driver. GreGetBoundsData is called by GpiQueryBoundaryData in response to an application's request for the current boundary data for a presentation space/device context pair. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	Option flags. See below.
pBoundsData	PRECTL	Pointer to the address for the returned RECTL structure that defines the specified bounding rectangle.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetBoundsData.

flOptions The option flags define which bounding rectangle the handling routine should return. Valid values are:

GBD_GPI Return GPI bounds in model space coordinates
GBD_USER Return current user bounds in screen coordinates and reset user bounds to their initial value.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetCodePage

```
#define INCL_GRE_DEVMISC3

LONG GreGetCodePage (hdc, pInstance, lFunction)
```

This function returns the current code page. This is the default code page obtained by WinQueryProcessCp during the enabling of the DC (page 7-12) or the code page set by GreSetCodePage. This function applies to the default font, not the currently selected font, which can be determined with a call to GreQueryFontAttributes (page 11-43).

Support: This function must be supported by the presentation driver. GreGetCodePage is called by GpiQueryCP in response to an application requesting the currently selected code page for the device context.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetCodePage

Return Codes: On completion, the handling routine must return the current code page (lCodePage) or GPI_ERROR.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. An error code for conditions that the handling routine is expected to check is:

PMERR_DEV_FUNC_NOT_INSTALLED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetCurrentPosition

```
#define INCL_GRE_LINES

BOOL GreGetCurrentPosition (hdc, pptlPosition, pInstance, lFunction)
```

This function takes the current (x, y) position in world coordinates from the DC instance data structure and stores it at the location addressed by pptlPosition. If COM_TRANSFORM is not set, the current position is returned in screen coordinates.

Support: This function must be supported by all presentation drivers. GreGetCurrentPosition is called by the function GpiQueryCurrentPosition. GreGetCurrentPosition might also be called in response to any of the Presentation Manager drawing functions that begin their operation from the current position within the presentation space.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pptlPosition	PPOINTL	Pointer to current position
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetCurrentPosition

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful.
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetDCOrigin

```
#define INCL_GRE_DEVMISC3

BOOL GreGetDCOrigin (hdc, pptlOrigin, pInstance, lFunction)
```

This function queries the origin of the device context that defines the bottom-left drawing origin for a display device or a banded hardcopy device. The DC origin is set by GreSetupDC (page 10-126) at the graphics engine, and by GreDeviceSetDCOrigin (page 8-50) at the presentation driver. This device context origin is stored in the Device Context Instance (DCI) data structure addressed by pInstance. The DC origin is always returned in screen coordinates.

Support: This function must be supported by presentation drivers for display devices and for hardcopy devices that use banding. For other devices, the minimum requirement is for the handling routine to return successful with pptlOrigin set to (0, 0).

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pptlOrigin	PPOINTL	Pointer to the (x, y) coordinates of the returned DC origin in screen coordinates
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetDCOrigin

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

GreGetLineOrigin

```
#define INCL_GRE_DEVMISC3

LONG GreGetLineOrigin (hdc, pptlXY, pInstance, lFunction)
```

This function returns the current line style from the DC instance data and stores the current position to the address indicated by pptlXY. The presentation driver maintains the line style information. Some lines and curves can be drawn either by the presentation driver or by simulations that must be able to query and set the style as required. If COM_TRANSFORM is not set, the coordinate pair at pptlXY is returned in screen coordinates.

The high-order WORD of the style number contains the first/last pel information. If the value of this byte is *0*, the first pel is not drawn. (See "First and Last Pel Considerations" on page 8-21.) The low-order byte indicates the position in the style mask. This is a count, held in the three least significant bits, of the number of positions that the style mask byte is rotated. The next byte is the state of the style error value.

Support: This function must be supported by the presentation driver. GreGetLineOrigin is used to get the line style and current position simultaneously. This function call can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pptlXY	PPOINTL	Pointer to an (x, y) coordinate pair to which the current position is returned
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetLineOrigin

Return Codes: On completion, this function returns the style number (lStyle), or GPI_ALTERROR if an error occurs.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetPairKerningTable

```
#define INCL_GRE_DEVMISC1

LONG GreGetPairKerningTable (hdc, cKernPairs, pKernPairs, pInstance, lFunction)
```

This function stores the kerning pairs of the current font to the buffer addressed by pKernPairs. The handling routine must transform all kerning-pair coordinates from device to world coordinates before sending the data to the calling routine. This can be done by using GreConvert. If it is unable to do this because the transform matrix is singular, it must log PMERR_COORDINATE_OVERFLOW.

Support: This function must be supported by the presentation driver. The call parameters are passed unchanged to the display driver's dispatch table. GreGetPairKerningTable is called from GpiQueryKerningPairs. GreGetPairKerningTable is used to return the kerning data for the currently selected font. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cKernPairs	LONG	Number of kerning pairs, requested by the application.
pKernPairs	PKERNPAIRS	Pointer to kern pair records. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetPairKerningTable.

pKernPairs KERNINGPAIRS structure:

sFirstChar	Code point for the first character
sSecondChar	Code point for the second character
sKerningAmount	2-byte signed integer indicating the amount of kerning. Positive numbers specify increased inter-character spacing.

Return Codes: On completion, the handling routine must return the number of kerning pairs returned in pKernPairs (cPairs), or GPI_ALTERROR if an error occurs.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SETID.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The number of kerning pairs is a field in the FONTMETRICS structure.

GreGetPel

```
#define INCL_GRE_BITMAPS

LONG GreGetPel (hdc, pptlPel, pInstance, lFunction)
```

This function returns the color of a pel at a specified position. If COM_TRANSFORM is set, this position is in world coordinates. If not set, the position is in screen coordinates. The return value of this function is either the color index of the pel or its RGB value depending on the color mode of the device.

Support: This function must be supported by the presentation driver, and is called by GpiQueryPel. GreGetPel is used to query the value of a pel at a specified (x, y) coordinate within the DC. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pptlPel	PPOINTL	Pointer to the coordinate (x, y) pair structure
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD=flags; low-order WORD=NGreGetPel

Return Codes: This function must return the color index value for the pel, or CLR_NOINDEX if there is no index corresponding to the color. In addition, the handling routine must raise an error when the point is subject to any clipping.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_PEL_IS_CLIPPED
PMERR_PEL_NOT_AVAILABLE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreImageData

```
#define INCL_GRE_BITMAPS

LONG GreImageData (hdc, pData, cBits, offRow, pInstance, lFunction)
```

This function draws a single row of image data with one bit per pel by using the current image foreground and background color and mix attributes. Drawing starts at the current x-position and at a y-position defined as a row offset below the current y-position. Data is supplied as a series of bytes and a count of the bits to be drawn. The handling routine cannot assume that unused bits at the end of the stream are set to *0*. Bits are drawn from left to right. The high-order bit of each byte is the leftmost-image bit. The 1 bits are foreground and the 0 bits are background. Notice that this function does not affect the current position.

Support: This function must be supported by the presentation driver. GreImageData is called by the function GpiImage. GreImageData is called multiple times for each call made to GpiImage, once for each scan line in the monochrome bit map.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pData	PBYTE	Pointer to data string.
cBits	LONG	Number of bits in row (maximum is 2040).
offRow	ULONG	Row number relative to the current y-position. Zero is the current position. A value of *1* indicates one row below the current position.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreImageData.

Return Codes: On completion, the handling routine must return a LONG integer (cHits), indicating whether correlation hits have been detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_IMAGE_DATA_LENGTH
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT

PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SCAN_START.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreLockDevice

```
#define INCL_GRE_DEVMISC3

BOOL GreLockDevice (hdc, pInstance, lFunction)
```

This function locks a device for use by a single thread.

Support: This function must be supported by all presentation drivers. GreLockDevice prevents two separate processes from accessing the resource (device context) at the same time. Hardcopy drivers need do nothing except return TRUE (Successful).

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreLockDevice

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function synchronizes the use and update of the visible region by allowing all current and pending drawings to finish and then blocking any requests to draw from other threads until GreUnlockDevice is called. On exit, the only thread allowed to continue with screen operations is the one that acquires the lock. To prevent deadlock, GreDeath cannot be called while the visible region is locked.

GreNotifyClipChange

```
#define INCL_GRE_DEVMISC2

BOOL GreNotifyClipChange (hdc, prclBound, cRect, idClipPath, pInstance, lFunction)
```

This function is called whenever there is any change to the DC region. The call gives the presentation driver an opportunity to optimize clipping by enumerating the clip rectangles and caching them whenever they change. Typically, the handling routine would allocate more memory for the new DC region (when necessary) and call GreGetClipRects (page 10-57) to get the set of rectangles that define the new DC region. Also (for display devices only), the handling routine must clear the HDC_IS_DIRTY flag. On completion, the handling routine must redispatch this call to the graphics engine using the pointer supplied in the default dispatch table.

Support: This function must be supported by the presentation driver. GreNotifyClipChange is called from the graphics engine whenever an application calls a GPI function that modifies the clipping rectangles within the device context. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclBound	PRECTL	Pointer to a rectangle that bounds the new region in screen coordinates.
cRect	ULONG	Number of rectangles in the new clip region as returned by GetClipRects.
idClipPath	ULONG	Current ClipPath ID. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreNotifyClipChange.

idClipPath For display devices, when idClipPath is passed as NCC_CLEANDC, the handling routine should clear the HDC_IS_DIRTY flag and return Successful. See "VisRegionNotify" on page 12-6 and "GreDeviceInvalidateVisRegion" on page 9-9.

Presentation drivers for other devices ignore this parameter.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Note: Presentation drivers that do not cache the clip rectangles should return TRUE, if there are no errors.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreNotifyTransformChange

```
#define INCL_GRE_DEVMISC2

BOOL GreNotifyTransformChange (hdc, flFlags, pXformdata, pInstance, lFunction)
```

This function notifies the presentation driver of a change in the transform from world coordinates to device coordinates. This function is called by the graphics engine to provide sufficient information to allow the device to optimize its calling of the GreConvert function or, where possible, to make all point transformations itself. The minimum requirement is for the handling routine to update the current position and pattern origin, and then call back to the GreNotifyTransformChange routine in the graphics engine. Notice that the handling routine must:

- Check that the new current position is valid.
- Check that the change will not cause an overflow of the 16-bit coordinates for the device space.
- Fail safe. If an error is detected or the call back to the graphics engine fails, the routine must restore the initial values before returning FALSE.

Support: This function must be supported by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flFlags	ULONG	See below.
pXformdata	PNOTIFYTRANSFORMDATA	Pointer to transform data structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreNotifyTransformChange.

flFlags A set of flags that pass information regarding the complexity of the 2x2 matrix of the M11, M12, M21, and M22 components (of the composite transform from world coordinates to device coordinates), and show if a translation is required. If the MATRIX_SIMPLE flag is not set, none of the other flags are valid.

MATRIX_SIMPLE	Two entries are *0*.
MATRIX_UNITS	All entries are *+1* or *−1*.
MATRIX_XY_EXCHANGE	Zeros are on the diagonal.
MATRIX_X_NEGATE	X is hit by negative (see Note).
MATRIX_Y_NEGATE	Y is hit by negative (see Note).
MATRIX_TRANSLATION	Non-zero translation.

Note: MATRIX_X_NEGATE and MATRIX_Y_NEGATE are only meaningful when both MATRIX_SIMPLE and MATRIX_UNITS are set.

Examples:

```
Matrix = { 1.0, 0.0, 0.0, 1.0, 0, 0 } =>
Flags = MATRIX_SIMPLE | MATRIX_UNITS

Matrix = { 1.0, 0.0, 0.0, 1.0, 5, 10 } =>
Flags = MATRIX_SIMPLE | MATRIX_UNITS | MATRIX_TRANSLATION

Matrix = { 0.0, -1.0, 1.0, 0.0, 17, 5 } =>
Flags = MATRIX_SIMPLE | MATRIX_UNITS | MATRIX_XY_EXCHANGE |
MATRIX_Y_NEGATE | MATRIX_TRANSLATION
```

pXformdata Pointer to NOTIFYTRANSFORMDATA structure:

usType	Indicates fixed-point notation
fxM11	Fixed-point matrix elements
fxM12	Fixed-point matrix elements
fxM21	Fixed-point matrix elements
fxM22	Fixed-point matrix elements
lM41	Long translations
lM42	Long translations

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PICK_APERTURE_POSN
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePolyLine

```
#define INCL_GRE_LINES

LONG GrePolyLine (hdc, paptlPoint, cPoints, pInstance, lFunction)
```

This function draws a sequence of one or more lines starting at the current (x, y) position. As each line is drawn, its end point becomes the start point for the next line. Upon completion, the current (x, y) position is the end point of the last line. When COM_TRANSFORM is not set, the function expects the array of points to be in screen coordinates.

When the COM_AREA or COM_PATH flag is set, this function is part of an area or path definition. In either case, the handling routine would usually pass the call back to the graphics engine for processing by the default handling routine. The call would not be passed back to the graphics engine if the presentation driver had hooked all of the area and path functions.

When this function is used to draw a closed figure, the handling routine must not draw the last point of the last (closure) line. The handling routine can call back to the graphics engine to do any necessary clipping.

Support: For performance reasons, all presentation drivers should support this function when drawing a polyline to a single clipping rectangle. When the clip region is more complex, the handling routine can forward the call to the graphics engine using the pointer supplied in the dispatch table when the presentation driver was enabled. The engine will clip each line and return it to the presentation driver as a call to GreDrawLinesInPath.

GrePolyLine is called by the function GpiPolyLine. GrePolyLine is also used by many of the complex object rendering routines within the graphics engine.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
paptlPoint	PPOINTL	Pointer to an array of (x, y) points. See below.
cPoints	LONG	Number of (x, y) pairs in points array
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePolyLine

cPoints When this is passed as *0*, the handling routine takes no action except to return Successful.

paptlPoint Pointer to an array of cPoints (x, y) pairs containing the (x, y) coordinates of the end points for the lines.

Return Codes: On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePolyMarker

```
#define INCL_GRE_MARKERS

LONG GrePolyMarker (hdc, paptlPoint, cPoints, pInstance, lFunction)
```

This function draws a sequence of one or more markers. The first marker is drawn at the current (x, y) position. Subsequent markers are drawn at the specified (x, y) positions that indicate the centers of the markers. Upon completion, the current (x, y) position is the center of the last marker. When COM_TRANSFORM is not set, the function expects the array of points to be in screen coordinates.

Support: This function must be supported by the presentation driver. GrePolyMarker is called by GpiPolyMarker and GpiMarker. GrePolyMarker is used to render one or more markers. The first marker is drawn at the current position and the positions of any subsequent markers must be specified.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paptlPoint	PPOINTL	Pointer to points array. See below.
cPoints	LONG	Number of markers to be drawn.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePolyMarker.

paptlPoint An array of (cPoints) (x, y) pairs that contain the (x, y) coordinates of the markers.
cPoints When this is passed as NULL, the handling routine takes no action except to return Successful.

Return Codes: On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_IN_AREA
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_PATH_LIMIT_EXCEEDED
PMERR_UNSUPPORTED_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePolyScanline

```
#define INCL_GRE_SCANS

LONG GrePolyScanline (hdc, pScanData, pInstance, lFunction)
```

This function fills an area lying between polyshortline pairs by using the current pattern attribute. Notice that coordinates are passed as unclipped screen coordinates. Filling is inclusive at the left boundaries and exclusive at the right boundaries. The scan lines are ordered from bottom-to-top and from left-to-right.

Support: This function must be supported by all presentation drivers *except* those that hook the GreDrawRLE, GreEndArea, and GreFillPath functions. All function calls to GrePolyScanline come from the GreDrawRLE, GreEndArea, or GreFillPath handling routines in the graphics engine. GrePolyScanline is not called by any specific function. However, it is likely to be accessed by any area-filling functions.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pScanData	PSCANDATA	Pointer to a SCANDATA structure
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePolyScanline

pScanData Pointer to a SCANDATA structure:

psllFirstLeft Pointer to the left end of the first polyshortline
psllLastLeft Pointer the left end of the last polyshortline
psllFirstRight Pointer to right edge of first polyshortline
psllLastRight Pointer to right edge of last polyshortline
c Number of scan lines
rclBound RECTL structure defining the bounding rectangle.

Notice that a *polyshortline* consists of a list of linked SHORTLINE structures:

slh SHORTLINEHEADER structure:

ulStyle Line style
ulFormat Line format
ptlStart (x, y) position of start
ptlStop (x, y) position of end
lxLeft Left edge of bounding rectangle
lxRight Right edge of bounding rectangle
pslhNext Pointer to next shortline
pslhPrev Pointer to previous shortline.

This structure is a discrete representation of a curve that starts at point (x0, y0) and ends at point (x1, y1). For each of the (y1-y0+1) rows, there is exactly one x value contained in the x array. The final point in the series is not drawn.

ax Array of *x* values as screen coordinates.

Return Codes: On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The handling routine can assume that the two polyshortlines do not cross and both polyshortlines have the same height.

GrePolyShortLine

```
#define INCL_GRE_LINES

LONG GrePolyShortLine (hdc, psl, pInstance, lFunction)
```

This function draws a series of shortlines. The current (x, y) position is not changed. A *polyshortline* is a linked list of shortlines. The function renders each SHORTLINE structure in the list until a NULL pslhNext is encountered. Notice that coordinates are passed as screen coordinates and are already completely clipped.

Support: This function must be supported by all presentation drivers except those that hook GrePolyLine and all of the GreArc*xxx* functions that are simulated by handling routines in the graphics engine. All function calls to GrePolyShortLine come from either GrePolyLine or the GreArc*xxx* functions. GrePolyShortLine might be accessed from any of the curve-rendering functions. However, it is not guaranteed that curve-rendering will call GrePolyShortLine.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
psl	PSHORTLINE	Pointer to SHORTLINE structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePolyShortLine.

psl The shortlines are defined in a list of linked SHORTLINE structures:

slh SHORTLINEHEADER structure:

ulStyle Line style.
ulFormat Line format.
ptlStart (x, y) position of start (the start position is included in the line).
ptlStop (x, y) position of end (the end position is not included in the line).
lxLeft Left edge of bounding rectangle.
lxRight Right edge of bounding rectangle.
pslhNext Pointer to next shortline.
pslhPrev Pointer to previous shortline.

Notice that the boundaries of the rectangle are inclusive at the start points of the lines, and exclusive at the stop points regardless of the direction.

This structure is a discrete representation of a curve that starts at point (x0, y0) and ends at point (x1, y1). For each of the (y1-y0+1) rows, there is exactly one x value contained in the x array. The final point in the series is not drawn.

ax Steps. This is an array of x-coordinates in absolute values. There is one coordinate value for each shortline.

Return Codes: On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryCharPositions

```
#define INCL_GRE_STRINGS

BOOL GreQueryCharPositions (hdc, pptlStart, flOptions, cChars, pchString, pAdx, paptXY, pInstance, lFunction)
```

This function stores at the location addressed by paptXY, an array of world coordinates identifying the start points at which the device is to place each character of a given string. GreQueryCharPositions is required for the management of device fonts in CM_MODE2 only. When the presentation driver has no device fonts, the handling routine must post PMERR_DEV_FUNC_NOT_INSTALLED.

Support: This function must be supported by the presentation driver. GreQueryCharPositions is called by GpiQueryCharStringPos, and is used to get the position where the next string output would occur relative to the current presentation space position. It also returns the starting position of each character within that string.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pptlStart	PPOINTL	Pointer to (x, y) coordinates of optional starting position.
flOptions	ULONG	Flags. See below.
cChars	LONG	Number of bytes in string.
pchString	PCH	Pointer to character string.
pAdx	PLONG	Pointer to Increment array. See below.
paptXY	PPOINTL	Pointer to array where character positions are returned.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryCharPositions.

flOptions The following flags can be set:

CHS_VECTOR If set, increment vector is present.
CHS_START_XY If set, starting position is present. Otherwise, pptlStart is ignored.

pAdx Vector of increment values with one element for each character in the string. After writing a character, the increment specifies the absolute distance in world coordinates to get to the starting point of the next character.

paptXY Pointer to an array of (cChars + 1) returned positions. The first value in the array is the initial current position; the last value is the current position on return.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_POS_OPTIONS
PMERR_INV_CODEPAGE
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_SETID
PMERR_INV_TRANSFORM_TYPE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryColorData

```
#define INCL_GRE_COLORTABLE

BOOL GreQueryColorData (hdc, cArray, pArray, pInstance, lFunction)
```

This function stores an array of information about the currently available logical color table and device colors at the location addressed by pArray. When the current table is the default logical color table, presentation drivers that support less than 16 colors return the device colors that the 16 colors from 0 (CLR_BACKGROUND) through 15 (CLR_PALEGRAY) have been mapped to.

Support: This function must be supported by the presentation driver. GreQueryColorData is called by GpiQueryColorData in response to an application's request for the currently selected color table data for the device context. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cArray	LONG	Number of elements supplied in Array.
pArray	PLONG	Pointer to Array. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryColorData.

pArray On return, this array contains:

lArray[QCD_LCT_FORMAT] Format of loaded color table (if any):

- **LCOLF_DEFAULT** Default color table is in force.
- **LCOLF_INDRGB** Color table loaded, which provides translation from index to RGB.
- **LCOLF_RGB** Color index = RGB.
- **LCOLF_PALETTE** DC has a palette selected.

lArray[QCD_LCT_LOINDEX] Smallest color index loaded (always *0*).

lArray[QCD_LCT_HIINDEX] Largest color index loaded (never less than *15*).

Information is only returned for the number of elements supplied, any extra elements supplied must be zeroed by the handling routine.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryColorIndex

```
#define INCL_GRE_COLORTABLE

LONG GreQueryColorIndex (hdc, flOptions, rgbColor, pInstance, lFunction)
```

This function returns the logical color index that is closest to the specified RGB color representation for the device. If the color index is RGB mode, the supplied RGB value is returned.

Support: This function must be supported by the presentation driver. GreQueryColorIndex is called by GpiQueryColorIndex when an application requests the index of the color most closely matching a specified color, relative to the current logical color table for the device context.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	See below.
rgbColor	LONG	Color, as an RGB value.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryColorIndex.

flOptions The only valid option is:

LCOLOPT_REALIZED If set, the information is required when the logical color table (if any) is realized. Otherwise, the information is required when the logical color table is not realized.

Other flags are reserved and must be *0*.

Return Codes: On completion, the handling routine must return the color index providing the closest match to the specified color (lColorIndex), or GPI_ALTERROR if an error occurred.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COLOR_OPTIONS
PMERR_INV_HDC
PMERR_INV_RGBCOLOR.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryDeviceBitmaps

```
#define INCL_GRE_DEVICE

BOOL GreQueryDeviceBitmaps (hdc, paOutData, cOutdata, pInstance, lFunction)
```

This function stores a list of bit-map formats supported by the device in the array addressed by paOutData. The number of formats supported can be found by using GreQueryDeviceCaps. Each format is returned in a pair of array elements and is in the form (cPlanes, cBitsPerPel). The first pair in the array must be the format that most closely matches the device.

Support: This function must be supported by the presentation driver. GreQueryDeviceBitmaps is called by GpiQueryDeviceBitmapFormats in response to the request of an application for a list of the bit-map formats that the device context supports. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paOutData	PLONG	Pointer to array where the bit-map format data is returned; excess elements are set to *0*.
cOutData	LONG	Number of elements in the array pointed to by paOutData. This must be even.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryDeviceBitmaps.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryDeviceCaps

```
#define INCL_GRE_DEVICE

BOOL GreQueryDeviceCaps (hdc, lIndex, paOutData, cOutData, pInstance, lFunction)
```

This function supports DevQueryCaps at the API. The handling routine returns the required information in the device capabilities buffer addressed by paOutData. Calls to GreQueryDeviceCaps would not usually require the handling routine to return data in all fields in the buffer. The parameters lIndex and cOutData identify the offset to the first field and the count of consecutive field for returned data.

Note: In OS/2 2.0, if GreQueryDeviceCaps returns data in the CAPS_DRIVER_VERSION field, the return value must be 00000200H.

Support: This function must be supported by the presentation driver. GreQueryDeviceCaps is called by DevQueryCaps, and is used to return information regarding the general capabilities of the device.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lIndex	LONG	Identifies the first item required. See below.
paOutData	PLONG	Pointer to an array where information is returned.
cOutData	LONG	Number of items of information to be returned at paOutData.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryDeviceCaps.

lIndex Indicates the element within the device capabilities array at which the presentation driver must begin returning information. Device capabilities are held by the system in an array of ULONG fields. For details, see DevQueryCaps in the *OS/2 2.0 Presentation Manager Programming Reference*.

Additional information is provided in the array for communication between the presentation driver and the graphics engine. The CAPS_ADDITIONAL_GRAPHICS field has two extra flags and a CAPS_DEVICE_FONT_SIM field is provided.

The additional flags in CAPS_ADDITIONAL_GRAPHICS are used (in conjunction with the CAPS_FONT_OUTLINE_DEFAULT and CAPS_FONT_IMAGE_DEFAULT flags) by the presentation driver when it wants the graphics engine to manage transforms and mappings for the default fonts that the driver supplies. The flags are:

CAPS_FONT_OUTLINE_MANAGE Set by the presentation driver to indicate that the graphics engine must manage the default outline font

CAPS_FONT_IMAGE_MANAGE Set by the presentation driver to indicate that the graphics engine must manage the default image font.

Note: If the presentation driver supplies the fonts but wants the graphics engine to manage them, it must pass the font address to the graphics engine using GreQueryDevResource.

The CAPS_DEVICE_FONT_SIM field contains flags that the presentation driver sets so that the graphics engine will handle simulations for the default fonts supplied by the driver:

CAPS_DEV_FONT_SIM_BOLD Indicates that the graphics engine should simulate CDEF_BOLD for device fonts

CAPS_DEV_FONT_SIM_ITALIC	Indicates that the graphics engine should simulate CDEF_ITALIC for device fonts
CAPS_DEV_FONT_SIM_UNDERSCORE	Indicates that the graphics engine should simulate CDEF_UNDERSCORE for device fonts
CAPS_DEV_FONT_SIM_STRIKEOUT	Indicates that the graphics engine should simulate CDEF_STRIKEOUT for device fonts.

Note: The font attributes CDEF_*xxx* are identified by the *cdef.fFlags* field in the character attributes bundle (see page 8-6). In the presentation driver, routines that write character strings should check the *cdef.fFlags* field to determine whether the call should be passed to the default handling routine in the graphics engine.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_LENGTH_OR._COUNT
PMERR_INV_QUERY_ELEMENT_NO.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryDevResource

```
#define INCL_GRE_DEVICE

LONG GreQueryDevResource (hdc, lType, id, pInstance, lFunction)
```

This function indicates whether a specified resource is available. If the resource is loaded, its handle is returned so that it can be selected into the device context. The resources (display information, pointers, bit maps, and fonts) are stored in DLL files. Some of these resources are linked by the presentation driver when it is first enabled, others are loaded by the application with WinLoadPointer.

The two system fonts are queried by the graphics engine when the presentation driver is loaded. When the presentation driver has a default font, it returns the handle of the font, as requested. When this function returns a NULL handle for the system font, the graphics engine default fonts are used instead (see "Font Functions" on page 11-1).

Support: This function must be supported by the display drivers, which must support the full range of requests. GreQueryDevResource is used internally by the graphics engine. Hardcopy drivers are required to provide a minimal level of support. At a minimum, the hardcopy driver must return *0* to indicate that a requested resource is not available. If a hardcopy driver has a raster or outline font that it requests the graphics engine to use as the default, then the presentation driver must return the address of its raster or outline font when the parameter, *id*, is equal to RT_FONT.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lType	ULONG	Resource type. See below.
id	ULONG	Defined resource value.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryDevResource.

lType Resource types returned by the presentation driver:

RT_DISPLAYINFO A structure containing some of the display constants required by the Window Manager. This information is required for all display devices that support windows. The format of the DISPLAYINFO structure is:

cb Size of this structure (always set to *26*).

cxIcon Count of pels for x-width of icon.

cyIcon Count of pels for y-height of icon. When WinLoadPointer is used to load the icon, it is stretched or compressed to the size indicated by cxIcon and cyIcon.

cxPointer Count of pels for x-width of pointer.

cyPointer Count of pels for y-height of pointer. When WinLoadPointer is used to load the pointer, it is stretched or compressed to the size indicated by cxPointer and cyPointer.

cxBorder Count of pels for x-width of horizontal border.

cyBorder Count of pels for y-height vertical border.

	cxHSlider	Count of pels for x-width of horizontal scroll bar slider.
	cyVSlider	Count of pels for y-height of vertical scroll bar slider.
	cxSizeBorder	Count of pels for x-width of default border.
	cySizeBorder	Count of pels for y-height of default border.
	cxDeviceAlign	Count of pels for horizontal device alignment.
	cyDeviceAlign	Count of pels for vertical device alignment. Some display devices operate faster when operation coordinates are aligned to a byte. Word or DWORD boundary. These two parameters allow the presentation driver to align windows on these boundaries and so optimize window management operations.
RT_POINTER	Defined system pointers are:	
	SPTR_ARROW	Left-pointing arrow, usually the system default.
	SPTR_TEXT	Text-insertion pointer, typically used when the mouse pointer is on an edit control.
	SPTR_WAIT	An hourglass used to tell the user to wait while the system is busy.
	SPTR_MOVE	Four arrows together, pointing north, south, east and west, that tell the user that window can be dragged in any of these directions.
	SPTR_SIZENWSE	An arrow pointing northwest and southeast, that tells the user that the window can be sized in these directions.
	SPTR_SIZENESW	An arrow pointing northeast and southwest, that tells the user that the window can be sized in these directions.
	SPTR_SIZEWE	An arrow pointing east and west, that tells the user that the window can be sized in these directions.
	SPTR_SIZENS	An arrow pointing north and south, that tells the user that the window can be sized in these directions.
	SPTR_APPICON	Usually a blank icon. This is used when a window that has been sized down to its minimum (and has no normal icon) is dragged across the screen.
	SPTR_ICONINFORMATION	Pointer used as part of a message box when specified in a call to WinMessageBox.
	SPTR_ICONQUESTION	Pointer used as part of a message box when specified in a call to WinMessageBox.
	SPTR_ICONERROR	Pointer used as part of a message box when specified in a call to WinMessageBox.
	SPTR_ICONWARNING	Pointer used as part of a message box when specified in a call to WinMessageBox.

	SPTR_ILLEGAL	Pointer used by the filing system to notify the user of an illegal mouse-directed copy or move operation.
	SPTR_MULTFILE	Pointer used by the file system to indicate a multiple file copy or move operation.
	SPTR_PROGRAM	Pointer used by the file system to indicate a copy or move operation on an executable program file.
	SPTR_FILE	Pointer used by the file system to indicate a copy or move operation on an ordinary file.
	SPTR_FOLDER	Pointer used by the file system to indicate a copy or move operation on an entire directory.
RT_BITMAP	The following defined system bit maps are required in display drivers:	
	SBMP_BTNCORNERS	Contains the rounded corners for pushbuttons. It is arranged as three bit maps divided into 2x2 bit map arrays describing the corners of each bit map. The three bit maps are defined as follows: • Contains the corners of an unselected pushbutton, which is not a default pushbutton. • Holds the corners of a default pushbutton that is currently not selected. • Contains the corners of a currently selected pushbutton, which can be either a default or non-default pushbutton.
	SBMP_DRIVE	Used by the file system to display the logical disk drive.
	SBMP_FILE	Used by the file system to indicate an unknown file type.
	SBMP_FOLDER	Used by the file system to display a directory.
	SBMP_MENUATTACHED	Drawn on the right edge of a menu item to indicate that a pulldown menu is attached to that item.
	SBMP_MENUCHECK	Displayed next to a menu item when the item is checked. Menu items are displayed in the system font and the menu checks are vertically aligned next to them. The height of this bit map must be no greater than the system font height to ensure that consecutive menu-check bit maps do not overlap. Its width is arbitrary, but is normally the same as the system font width.
	SBMP_PROGRAM	Used by the file system to mark EXE and COM files.
	SBMP_SIZEBOX	Used by some applications to display a *sizebox* in the bottom-right corner of a frame window.

SBMP_TREEMINUS Used by the file system to indicate there are no more subdirectories to view.

SBMP_TREEPLUS Used by the file system to indicate there are more subdirectories to view.

The following defined system bit maps are also required to ensure compatibility with OS/2 Version 1.x applications:

SBMP_OLD_CHILDSYSMENU
SBMP_OLD_CHECKBOXES
SBMP_OLD_MAXBUTTON
SBMP_OLD_MINBUTTON
SBMP_OLD_RESTOREBUTTON
SBMP_OLD_SBDNARROW
SBMP_OLD_SBLFARROW
SBMP_OLD_SBRGARROW
SBMP_OLD_SBUPARROW
SBMP_OLD_SYSMENU.

The following defined system bit maps are provided by PMWIN.DLL and are *optional* in display drivers:

SBMP_CHECKBOXES
SBMP_CHILDSYSMENU
SBMP_CHILDSYSMENUDEP
SBMP_CLOSEBUTTON
SBMP_CLOSEBUTTONDEP
SBMP_COMBODOWN
SBMP_MAXBUTTON
SBMP_MAXBUTTONDEP
SBMP_MINBUTTON
SBMP_MINBUTTONDEP
SBMP_RESTOREBUTTON
SBMP_RESTOREBUTTONDEP
SBMP_SBDNARROW
SBMP_SBDNARROWDEP
SBMP_SBDNARROWDIS
SBMP_SBLFARROW
SBMP_SBLFARROWDEP
SBMP_SBLFARROWDIS
SBMP_SBRGARROWDEP
SBMP_SBRGARROWDIS
SBMP_SBRGARRROW
SBMP_SBUPARROW
SBMP_SBUPARROWDEP
SBMP_SBUPARROWDIS
SBMP_SYSMENU
SBMP_SYSMENUDEP.

Refer to the function WinGetSysBitmap in *OS/2 2.0 Presentation Manager Programming Reference* for more information.

RT_FONT The default system fonts are:

SFONT_RASTER Default image font
SFONT_OUTLINE Default outline font.

Return Codes: This function returns the address of the indicated resource, *0* if no address is available, or GPI_ALTERROR if an error occurs.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_DEV_FUNC_NOT_INSTALLED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryHardcopyCaps (Hardcopy Drivers Only)

```
#define INCL_GRE_DEVICE

LONG GreQueryHardcopyCaps (hdc, lStart, cCount, pInfo, pInstance, lFunction)
```

This function stores information about the hardcopy capabilities of the device in the buffer addressed by pInfo. The information is stored as a sequence of one or more HCINFO structures defining the hardcopy capabilities for one or more form codes. For presentation drivers that support more than one form code with the relevant data held in a structure of HCINFO structures, the parameters lStart and cCount identify the starting point in the main structure and the number of HCINFO structures to be stored.

It is usual for this function to be issued twice (initially with a value of *0* in cCount) to return the number of forms available. Storage can be allocated and the function called again with cCount set to the number of forms for which information is required.

Support: This function must be supported by hardcopy drivers. It is not required for display drivers. GreQueryHardcopyCaps is called by the function DevQueryHardcopyCaps.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lStart	LONG	Index. See below.
cCount	LONG	Number of forms. See below.
pInfo	PHCINFO	Pointer to buffer for returned form data. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryHardcopyCaps.

lStart Index to the required starting HCINFO structure. A value of *0* identifies the HCINFO for the first form.

cCount Number of structures to be returned in the buffer. A value of *0* requests the handling routine to set the return code to the number of forms that the driver supports.

pInfo A pointer to the buffer for the returned data. The data is returned as a set of one or more HCINFO structures. When pInfo is not equal to NULL and lStart is greater than, or equal to, the number of form codes that the hardcopy drivers supports, the hardcopy driver should return *0* without modifying the memory block pointed to by pInfo.

szFormname[32]	Character string containing the name of the form
cx	Width (left-to-right) in millimeters
cy	Height (top-to-bottom) in millimeters
xLeftClip	Left clip limit in millimeters
yBottomClip	Bottom clip limit in millimeters
xRightClip	Right clip limit in millimeters
yTopClip	Top clip limit in millimeters
xPels	Number of pels between left and right clip limits
yPels	Number of pels between bottom and top clip limits
flAttributes	Attributes describing the availability of the form:

HCAPS_CURRENT	The form is currently available on the device. For devices with multiple paper trays, HCAPS_CURRENT says that the paper required for this form is in the current paper tray.
HCAPS_SELECTABLE	The form is installed on the device but has to be selected. That is, the paper tray required for this form is not the current one.

Return Codes: The value returned by the handling routine depends on the initial value of cCount. If cCount=0, return the total number of form codes that the presentation driver supports. For any other value, return the number of HCINFO structures that were transferred to the buffer.

Possible Errors Detected: When an error is detected, the handling routine must return DQHC_ERROR and call WinSetErrorInfo to post the condition. An error code for conditions that the handling routine is expected to check is:

PMERR_INV_LENGTH_OR._COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryLogColorTable

```
#define INCL_GRE_COLORTABLE

LONG GreQueryLogColorTable (hdc, flOptions, lStart, cArray, pArray, pInstance, lFunction)
```

This function stores an array of the current logical color values at the location addressed by pArray.

Support: This function must be supported by the presentation driver. GreQueryLogColorTable is called by GpiQueryLogColorTable in response to an application's request for the currently configured logical color table. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	See below.
lStart	LONG	Starting index for which data is to be returned.
cArray	LONG	Number of elements available in the array.
pArray	PLONG	Pointer to the array in which the information is returned.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryLogColorTable.

flOptions The only valid option flag is:

LCOLOPT_INDEX If set, the handling routine must return the index for each RGB value.

pArray When LCOLOPT_INDEX is set, pArray points to an array of alternating color indexes and values (index1, value1, index2, value2, and so forth). If the logical color table is not loaded with a contiguous set of indexes, any index values that are not loaded are skipped.

When LCOLOPT_INDEX is not set, pArray points to an array of RGB color values in which the information is to be returned. Each value is the same as those defined for "GreCreateLogColorTable" on page 8-34, starting with the specified index and ending when there are no further loaded entries in the table or when cCount has been exhausted. If the logical color table is not loaded with a contiguous set of indexes, QLCT_NOTLOADED is returned as the color value for any index values that are not loaded.

Return Codes: The handling routine must return a LONG value indicating the number of elements returned in pArray or:

QLCT_ERROR Error
QLCT_RGB Color table is in RGB mode and no elements are returned.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryNearestColor

```
#define INCL_GRE_COLORTABLE

LONG GreQueryNearestColor (hdc, flOptions, rgbColorin, pInstance, lFunction)
```

This function returns the available color nearest to the specified color on the currently associated device even if it is not available in the logical color table. Both colors are specified as RGB values. The color used for drawing primitives such as lines and text is the color returned. GreQueryNearestColor does not consider the possibility of using *dithered* colors for filling areas. Where dithered colors are used for filling, the color used for text and lines is likely to be different even when the same color index is selected.

The nearest color is determined by finding its position in RGB space. *RGB space* can be defined as a cube with three axes (representing red, green and blue color intensities) radiating from one corner or origin. Moving up the Red axis results in increasing red intensity. Different intensities of cyan can be produced by moving along the Green and Blue axes. For EGA and VGA devices, dithering is performed by dividing the RGB space into 9x9x9 cubical cells representing the colors that can be created. The cell each RGB color falls into is determined and a lookup table is created to indicate which EGA planes to set *on* or *off* to create each color.

When this function is called for a monochrome device, the color returned is either the reset color or the contrast color for the device. See "Support for Monochrome Devices" on page 8-15.

Support: This function must be supported by the presentation driver. GreQueryNearestColor is called by GpiQueryNearestColor when the application wants the available colors nearest to the specified color.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	See below.
rgbColorIn	LONG	Color required.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryNearestColor.

flOptions The only significant flag is:

LCOLOPT_REALIZED If set, the information is required when the logical color table (if any) is realized. When this flag is not set, the information is required when the logical color table is not realized.

Other flags are reserved.

Return Codes: The handling routine must return the nearest available RGB color to that requested (rgbColorOut), or GPI_ALTERROR if an error occurred.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COLOR_OPTIONS
PMERR_INV_HDC
PMERR_INV_RGBCOLOR.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryRealColors

```
#define INCL_GRE_COLORTABLE

LONG GreQueryRealColors (hdc, flOptions, lStart, cArray, pArray, pInstance, lFunction)
```

This function stores, in the array addressed by pArray, the RGB values of the distinct colors available on the currently associated device.

Support: This function must be supported by the presentation driver. GreQueryRealColors is called by GpiQueryRealColors in response to an application requesting the currently available colors for the device context.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	See below.
lStart	LONG	Ordinal number of the first color required. See below.
cArray	LONG	Number of elements available in the array.
pArray	PLONG	Pointer to array in which data is returned.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryRealColors.

flOptions Valid options are:

LCOLOPT_REALIZED If set, the information is required when the logical color table (if any) is realized. When this flag is not set, the information is required when the logical color table is not realized.

LCOLOPT_INDEX If set, the handling routine must return the index for each RGB value. Other flags are reserved and must be *0*.

lStart Typically, this is *0* to start the sequence. This value does not necessarily bear any relationship to the color index because the order in which the colors are returned is not defined.

pArray When LCOLOPT_INDEX is set, this is an array of alternating color indexes and values (in the order, index1, value1, index2, value2 and so forth). If there is a color table, colors that are not in the table but are available on the device, have a special index of QLCT_NOTLOADED. In RGB mode, the RGB value is returned in the color indexes.

When LCOLOPT_INDEX is not set, this is an array of color values. Each value is the same as those defined for "GreCreateLogColorTable" on page 8-34.

Return Codes: On completion, the handling routine must return the number of colors returned in the array (cColors), or GPI_ALTERROR if an error occurred

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryRGBColor

```
#define INCL_GRE_COLORTABLE

LONG GreQueryRGBColor (hdc, flOptions, iColor, pInstance, lFunction)
```

This function returns the actual RGB color that results from the specified color index for the specified device If the color index is RGB mode, the nearest RGB color (the same as QueryNearestColor) is returned. All defined indexes are valid except CLR_DEFAULT, which causes an error to be raised.

Support: This function must be supported by the presentation driver. GreQueryRGBColor is called by GpiQueryRGBColor in response to the request of an application to convert a color index into the corresponding RGB value. If the logical table is currently in RGB mode, the nearest RGB color is returned. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	See below.
iColor	LONG	Color index.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryRGBColor.

flOptions Valid options are:

LCOLOPT_REALIZED If set, the information is required when the logical color table (if any) is realized. When this flag is not set, the information is required when the logical color table is not realized.

LCOLOPT_INDEX When set, the handling routine must return the actual RGB color originally specified for this index. Otherwise, it must return the nearest RGB color for this index, that is, the one which would result from drawing on the specified device.

Other flags are reserved and must be *0*.

Return Codes: On completion, the handling routine must return the nearest available RGB color to that requested (rgbColor), or GPI_ALTERROR if an error occurred.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COLOR_OPTIONS
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryTextBox

```
#define INCL_GRE_STRINGS

BOOL GreQueryTextBox (hdc, cChars, pchString, cptPosition, paptPosition, pInstance, lFunction)
```

This function processes a character string as if it were being drawn. The handling routine stores the coordinates of the current text box (relative to the current (x, y) position) as an array at the location indicated by paptPosition. The first four coordinate pairs identify the bounding parallelogram for the given character string. The fifth coordinate pair is the (x, y) position of the starting point for the next character position after the string, that is, the current position value that would be set by an equivalent call to GpiCharStringAt. The positions take account of current values for character spacing such as kerning and character space. The points on the borders of the character box are deemed to be inside the box.

When the character mode is CM_MODE2, this function is valid only if the character angle and shear attributes are set to their default values. See "Character Attributes" on page 8-6.

Support: This function must be supported by the presentation driver. GreQueryTextBox is called by the function GpiQueryTextBox, and is used to return a tight bounding box for the currently selected font of a given string relative to the current position.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cChars	LONG	Number of bytes in string.
pchString	PCH	Pointer to character string.
cptPosition	LONG	Number of (x, y) pairs that the Position array can contain.
paptPosition	PPOINTL	Pointer to position array. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryTextBox.

paptPosition Pointer to the array in which the function returns the requested values. Upon completion, the array contains cptPosition sets of (x, y) coordinates in the following order:

TXTBOX_TOPLEFT	Top-left corner
TXTBOX_BOTTOMLEFT	Bottom-left corner
TXTBOX_TOPRIGHT	Top-right corner
TXTBOX_BOTTOMRIGHT	Bottom-right corner
TXTBOX_CONCAT	Start point of next character position.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY

PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_SETID
PMERR_INV_TRANSFORM_TYPE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryWidthTable

```
#define INCL_GRE_STRINGS

BOOL GreQueryWidthTable (hdc, lFirstChar, cWidthTable, paWidthTable, pInstance, lFunction )
```

This function stores, at the location addressed by paWidthTable, an array of world coordinates representing the width table information of the currently selected font. The handling routine must use GreConvert (page 10-26) to transform the width-table information from device coordinates to world coordinates.

Support: This function must be supported by the presentation driver. GreQueryWidthTable is called by the function GpiQueryWidthTable. GreQueryWidthTable returns an array of world coordinates representing the width-table information for the currently selected font.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lFirstChar	LONG	Code point of the initial character for which width-table information is required
cWidthTable	LONG	Count of widths in the width-table data
paWidthTable	PLONG	Pointer to buffer of width-table data
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryWidthTable

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COORD_SPACE
PMERR_INV_FIRST_CHAR
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_SETID
PMERR_INV_TRANSFORM_TYPE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreRealizeColorTable

```
#define INCL_GRE_COLORTABLE

BOOL GreRealizeColorTable (hdc, pInstance, lFunction)
```

Note: GreRealizeColorTable is provided for the support of older applications and is of less importance for new applications.

This function causes the system to ensure that, for a realizable color table, the device physical color table is set to the closest possible match to the logical color table. A device context such as a hardcopy DC can implicitly realize a color table when it is created. In this case, the handling routine need only return Successful. An error must be posted if this function is called for a color table that cannot be realized.

Support: This function must be supported by the presentation driver. GreRealizeColorTable is called by GpiRealizeColorTable in response to the request of an application to realize the current logical color table to device output capabilities. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRealizeColorTable

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COL_TABLE_NOT_REALIZABLE
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_REALIZE_NOT_SUPPORTED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreRealizeFont

```
#define INCL_GRE_DEVMISC2

ULONG GreRealizeFont (hdc, cmdCommand, pLogFont, pFont, pInstance, lFunction)
```

This function requests the presentation driver to realize or delete a font.

Support: This function must be supported by the presentation driver. GreRealizeFont is called during the processing of GpiCreateLogFont to realize the specified logical font. This function call can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cmdCommand	ULONG	See below.
pLogFont	PFATTRS	Pointer to a logical font. See below.
pFont	PULONG	See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRealizeFont.

cmdCommand Valid commands are:

RF_DEVICE_FONT (1). The graphics engine asks the presentation driver if it is able to realize a device font for the logical font identified by pLogFont. The action taken by the handling routine depends on the match number of the logical font. The match number is the lMatch parameter in the FONTMETRICS structure of the font specified. Font matching is discussed on page 11-1.

- If lMatch is negative, the request is for the device font with the corresponding match number. The handling routine must return the font handle, or *0* if no match was found.
- If lMatch is *0*, the handling routine must search the device fonts for an exact match to pLogFont and return its handle, or *0* if no match was found.
- If lMatch is a positive value, the handling routine must return *0* as if no font was found.

RF_LOAD_ENGINE_FONT (2). The presentation driver is requested to convert an engine font into a device font. The presentation driver must return the font handle, or *0* if it is unable to convert the font.

RF_DELETE_FONT (3). The presentation driver is requested to delete a device font. The 32-bit font handle is passed in pFont (see page 8-131).

RF_DELETE_ENGINE_FONT (4). Informs the presentation driver that a previously loaded engine font is being deleted. If font caching is not supported, the handling routine performs action.

The font is passed to the presentation driver in the device character bundle during the GreDeviceSetAttribute call. (See "Character Attributes" on page 8-6.) A bit in the CHARDEFS structure indicates whether this is a pointer to an engine font or the handle to a device font.

pLogfont The initial value of cmdCommand determines the nature of this parameter:

RF_DEVICE_FONT	Pointer to a logical font
RF_LOAD_ENGINE_FONT	Pointer to a logical font
RF_DELETE_FONT	NULL pointer
RF_DELETE_ENGINE_FONT	NULL pointer

pFont The initial value of cmdCommand determines the nature of this parameter:

RF_DEVICE_FONT	NULL pointer
RF_LOAD_ENGINE_FONT	Pointer to an engine font
RF_DELETE_FONT	32-bit device font handle
RF_DELETE_ENGINE_FONT	Pointer to an engine font.

Return Codes: The value returned by this function depends on whether the presentation driver is requested to realize or to delete the font. When realizing or loading a font, the handling routine must return a 32-bit logical font handle (GPI_ALTERROR), or *0* if the match (or load) was unsuccessful.

When deleting a font, the handling routine must return:

GPI_OK	Successful
GPI_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: GreRealizeFont is called by the graphics engine to allow the presentation driver to satisfy logical font requests. The example below shows a typical font-matching algorithm for the presentation driver:

```
/* If the face name is empty, the application is requesting the default font.  Return NO_MATCH.*/

   if(Font->szFacename = NULL)
   {
      return(0);
   }
```

```
/* If the match number is positive, an engine font is required.  Return NO_MATCH.          */

   if(Font->lfMatch > 0)
   {
      return(0);
   }

/* If the match number is negative, a device font is required.  The presentation driver should */
/* return the font if it exists; otherwise return NO_MATCH.                                    */

   if(Font->lMatch < 0)
   {
     if(font with required lMatch exists)
     {
     /* Check the fsSelection flags and szFacename.  If the font is unable to satisfy these    */
     /* flags, return NO_MATCH.  Otherwise, return the device font handle.                      */

       if((FATTR_FONTUSE_OUTLINE && font not outline)
         ||(FATTR_FONTUSE_TRANSFORMABLE && font not outline) ||(szFacename does not match font))
          return(0);
       else
          return(device_font_handle);
     }
     else
     {
     /* Attempt metrics match (i.e. all metrics including szFaceName, usCodePage, lAveCharWidth */
     /* and lMaxBaselineExt).  If match exists, return device font handle.  Otherwise, return   */
     /* NO_MATCH.                                                                               */

       if(metric match exists)
       {
          return(device_font_handle);
       }
       else
          return(0);
     }
   }

/* The match number is zero, the presentation driver should search for a font with the         */
/* specified metrics, and if an exact match exists, return the device font handle. Otherwise,  */
/* return NO_MATCH.                                                                            */

if(Font->lMatch = 0)
{
  if(metrics match exists - see above)
  {
  /* Check the fsSelection flags.  If the font is unable to satisfy these flags, return        */
  /* NO_MATCH.  Otherwise, return the device font handle.                                      */

    if((FATTR_FONTUSE_OUTLINE && font not outline)
    ||(FATTR_FONTUSE_TRANSFORMABLE && font not outline)
       return(0);
    else
       return(device_font_handle);
  }
else
   return(0);
}
```

GreResetBounds

```
#define INCL_GRE_DEVMISC3

BOOL GreResetBounds (hdc, flOptions, pInstance, lFunction)
```

This function resets the bounds to their initial values, 07FFFFFH for the minimum coordinates and F800000H for the maximum coordinates. Hardcopy drivers are required to support only GPI bounds. Display drivers must also support user bounds for the Window Manager.

Support: This function must be supported by the presentation driver. GreResetBounds is called by the function GpiResetBoundaryData, and is used to reset the boundary data for a presentation space or device context pair.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	Option flags. See below.
pInstance.	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreResetBounds.

flOptions Valid flags are:

RB_GPI Reset the GPI bounds
RB_USER Reset the user bounds

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetBitmapBits

```
#define INCL_GRE_BITMAPS

LONG GreSetBitmapBits (hdc, hbm, lScanStart, cScanCount, pBitmap, pInfo, pInstance, lFunction)
```

This function transfers bit-map data from application storage into the specified bit map or DC. The bit map can be specified by its handle, or (if this is NULL) a DC handle. In this case, the device context must be a memory DC with a bit map currently selected. This function does not set bits directly into any other kind of device. When the format of the supplied bit map does not match that of the device, the handling routine must convert it using the supplied BITMAPINFO or BITMAPINFO2 structure. Only standard formats and device formats that are compatible with the target device are supported.

Support: This function must be supported by the presentation driver. GreSetBitmapBits is called from the function GpiSetBitmapBits, and is used to transfer bit-map data from the device context to application storage. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hbm	HBITMAP	Bit-map handle.
lScanStart	LONG	Scan line number from where data transfer starts; *0* is the first.
cScanCount	LONG	Number of scan lines to be transferred.
pBitmap	PBYTE	Pointer to bit-map data. See below.
pInfo	PBITMAPINFO	Pointer to BITMAPINFO or BITMAPINFO2 structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetBitmapBits.

pBitmap Pointer to the pel data of the bit map. This data is stored in the order that the coordinates appear on a display screen, that is, the pel in the lower-left corner is the first in the bit map. Pels are scanned to the right, and upward, from that position. The bits of the first pel are stored beginning with the most significant bits of the first byte. The data for pels in each scan line is packed together tightly. However, all scan lines are padded at the end so that each one begins on a ULONG boundary. That is, three bytes of pel data will hold one 24-bit pel, three 8-bit pels, six 4-bit pels, or twenty-four 1-bit pels. If those three bytes are the only pel data for that scan line, one more byte of zeros would be required to pad the line to a ULONG boundary.

pInfo Pointer to either a BITMAPINFO structure:

cbFix	Length of structure
cx	Bit-map width
cy	Bit-map height
cPlanes	Number of color planes, *1* if standard format
cBitCount	Number of adjacent color bits per pel
argbColor[]	Color table array of RGB structures:
	bBlue
	bGreen
	bRed.

Or pointer to a BITMAPINFO2 structure:

cbFix	Length of structure

cx Bit-map width

cy Bit-map height

cPlanes Number of planes, *1* if standard format

cBitCount Number of adjacent color bits per pel

ulCompression Compression scheme used to store the bit map:

BCA_UNCOMP Bit map is uncompressed (the only valid value).

cbImage Length of bit-map storage data in bytes. If the bit map is uncompressed, *0* (default) can be specified for this.

cxResolution Horizontal component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution Vertical component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cclrUsed The number of color indexes from the color table that are used by the bit map. If it is *0* (default), all the indexes are used. If it is non-zero, only the first cclrUsed entries in the table are accessed by the system; further entries can be omitted.

For standard formats with a cBitCount of 1, 4, or 8 (and cPlanes = 1), any indexes beyond cclrUsed are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, cclrUsed is the number of colors used by the bit map.

cclrImportant Minimum number of colors indexes for satisfactory appearance of the bit map. More colors can be used in the bit map but it is not necessary to assign them to the device palette. These additional colors can be mapped to the nearest colors available. Zero (default) means that all entries are important. For a 24-bitcount standard format, the cclrImportant colors are also listed in the color table relating to this bit map.

usUnits Units of measure of the horizontal and vertical components of resolution:

BRU_METRIC (Default.) Pels per meter.

usReserved Reserved field. If present, it must be *0*.

usRecording Recording algorithm, the format in which bit-map data is recorded:

BRA_BOTTOMUP (Default.) Scan lines are recorded from bottom-to-top.

usRendering Half-toning algorithm used to record bit-map data that has been digitally half-toned:

BRH_NOTHALFTONED	(Default.) Bit-map data not half-toned.
BRH_ERRORDIFFUSION	Error diffusion or damped error diffusion algorithm
BRH_PANDA	Processing algorithm for noncoded document acquisition
BRH_SUPERCIRCLE	Super circle algorithm

cSize1 Size value 1. If BRH_ERRORDIFFUSION is specified in usRendering, cSize1 is the error damping as a percentage in the range 0 – 100. A value of 100% indicates no damping. A value of 0% indicates that any errors are not diffused.

If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize1 is the x-dimension of the pattern used in pels.

cSize2 Size value 2. If BRH_ERRORDIFFUSION is specified in usRendering, this parameter is ignored. If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize2 is the y-dimension of the pattern used in pels.

ulColorEncoding Color encoding:

BCE_RGB (Default.) Each element in the color array is an RGB2 data type.

ulIdentifier Reserved for application use.

argbColor[] Color table array of RGB2 structures:

bBlue
bGreen
bRed
fcOptions Reserved. Must be *0*.

Return Codes: On completion, the handling routine must return a LONG value (lLines), indicating the number of lines transferred, or GPI_ALTERROR if an error occurred.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INCORRECT_DC_TYPE
PMERR_INV_HBITMAP
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SCAN_START.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When the bit-map handle is NULL, the DC must be a memory DC with a bit map currently selected. Otherwise, the DC must be valid for that device. When the format of the supplied bit map does not match that of the device, the handling routine must use the supplied BITMAPINFO or BITMAPINFO2 structure to convert it.

GreSetCodePage

```
#define INCL_GRE_DEVMISC3

BOOL GreSetCodePage (hdc, lCodePage, pInstance, lFunction)
```

This function sets the current code page for characters written with the base (default) font. The default is the font that the system uses when the cbnd.usSet attribute is 0000H. (See "Character Attributes" on page 8-6.) When the base font is not in use, lCodePage is saved until required. When a DC is initialized, the code page is set to the default code page obtained from WinQueryProcessCp.

Support: This function must be supported by the presentation driver. GreSetCodePage is called by GpiSetCP in response to an application requesting to change the currently selected code page for the device context.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
lCodePage	ULONG	New code page
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetCodePage

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_INV_HDC
PMERR_INV_IN_AREA.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetCurrentPosition

```
#define INCL_GRE_LINES

BOOL GreSetCurrentPosition (hdc, pptlPosition, pInstance, lFunction)
```

This function sets the current (x, y) position and resets the line type sequence. Typically, the handling routine also sets a flag in the DC instance data structure to indicate that the first pel of the next line must be drawn. When the COM_AREA or COM_PATH flag is set, this function is part of an area or path definition. In either case, the handling routine usually passes the call back to the graphics engine for processing by the default handling routine. The call would not be passed back to the graphics engine if the presentation driver had hooked all of the area and path functions.

Support: This function must be supported by all presentation drivers. GreSetCurrentPosition is called by the function GpiSetCurrentPosition. GreSetCurrentPosition might be called in response to any of the Presentation Manager drawing functions, which end their operations by updating the current presentation space position.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pptlPosition	PPOINTL	Pointer to new current position. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetCurrentPosition.

pptlPosition When COM_TRANSFORM is set, the current position is expressed in world coordinates. Otherwise, this value is passed in device coordinates. The handling routine must transform these values as appropriate. Typically, the presentation driver maintains the current position in both coordinate sets in the DC instance data structure.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_PATH_LIMIT_EXCEEDED.

Remarks: When the current context is *in area*, a figure closure line is generated (if necessary), which can cause a correlation hit to occur on an area boundary. The current position should only be correlated on, merged into the bounds, or both correlated and merged, if it is actually used in a drawing primitive.

The following is an example:

```
            ...
GreSetCurrentPosition (hdc, p1);
            ...
            ...
            ...
GreSetCurrentPosition (hdc, p2);
            ...
            ...
            ...
GrePolyLine (hdc, to p3, n);
            ...
```

Notice that the sequence does not merge p1 into the bounds or correlate on it.

GreSetLineOrigin

```
#define INCL_GRE_DEVMISC3

BOOL GreSetLineOrigin (hdc, pptlXY, lStyle, pInstance, lFunction)
```

This function sets the current line style and current position. If COM_TRANSFORM is set, the current position is expected in world coordinates. If COM_TRANSFORM is not set, the current position is expected in screen coordinates. The new line style is stored in the DC instance data structure. Some lines and curves can be drawn either by the presentation driver or by simulations, and therefore must be able to query and set the style as required.

The high-order WORD of the style number contains the first/last pel information. The low-order byte indicates the position in the style mask. The next byte is the state of the style error value. See "GreGetLineOrigin" on page 8-90.

Support: This function must be supported by the presentation driver. GreSetLineOrigin is used to enable the simultaneous update of line style and position. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pptlXY	PPOINTL	Pointer to an (x, y) coordinate pair to which the current position is returned
lStyle	ULONG	Style number
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetLineOrigin

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The C Language example that follows outlines a strategy by which the handling routine could use the information from a call to GreSetLineOrigin.

```
/*****************************************************************/
/*                                                               */
/*   StyleMask is a single byte.  usState is a USHORT with the   */
/*   error byte as the low-order byte and the mask position as   */
/*   the high-order byte.  The three low-order bytes of the mask */
/*   position represent the number of bits by which StyleMask has */
/*   been rotated.                                               */
/*                                                               */
/*****************************************************************/

    while (necessary--)

          {

/*****************************************************************/
/*                                                               */
/*   Do we need to draw the first pel in the line?               */
/*                                                               */
/*****************************************************************/

          if (stylemask & 0x80)
              SetPel (x, y);

/*****************************************************************/
/*                                                               */
/*   Save the current style state.                               */
/*                                                               */
/*****************************************************************/

          usStateOld=usState;
          switch (LineMajor)
              {
              case yMajor:
                   usState=usState+yRatio;
                   break;
              default:
                   usState=usState+xRatio;
                   break;
              }

           if HIBYTE (usState) != HIBYTE (usStateOld)

/*****************************************************************/
/*                                                               */
/*   If the error byte has overflowed, rotate the style ratio.   */
/*   The style mask is reset every eighth rotation.              */
/*                                                               */
/*****************************************************************/

             RotateLeftOne (StyleRatio);
             UpDateNext (x, y);
          }

/*****************************************************************/
/*                                                               */
/*****************************************************************/
```

The values for pRatio are set and queried by SetStyleRatio and GetStyleRatio respectively (see pages 9-21 and 9-13). Alternatively, the programmer can specify device-specific defaults for xRatio and yRatio.

GreSetPel

```
#define INCL_GRE_BITMAPS

LONG GreSetPel (hdc, pptlPel, pInstance, lFunction)
```

This function sets a pel to the current line attribute, color and mix. If COM_TRANSFORM is set, the pel position is expected in world coordinates. If COM_TRANSFORM is not set, the pel position is expected in screen coordinates. This function is subject to all usual clipping. No error is returned when the point is clipped.

Support: This function must be supported by the presentation driver. GreSetPel is called by the function GpiSetPel, and is used to set the value of a pel at a specified (x, y) coordinate within a device context. This function can be handled by bit map-simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pptlPel	PPOINTL	Pointer to pel position in world or screen coordinates
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetPel

Return Codes: On completion, the handling routine must return a LONG integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by the display driver when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreUnlockDevice

```
#define INCL_GRE_DEVMISC3

BOOL GreUnlockDevice (hdc, pInstance, lFunction)
```

This function allows all pending screen input or output operations blocked by GreLockDevice to continue.

Support: This function must be supported by all presentation drivers. For hardcopy devices, the hardcopy driver need do nothing except return TRUE (Successful). GreUnlockDevice is used to enable another process to access a resource (device context) that had been previously locked to prevent simultaneous update.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreUnlockDevice

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. An error code for conditions that the handling routine is expected to check is:

PMERR_DEV_FUNC_NOT_INSTALLED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function is used to synchronize the use and update of the visible region.

GreUnrealizeColorTable

```
#define INCL_GRE_COLORTABLE

BOOL GreUnrealizeColorTable (hdc, pInstance, lFunction)
```

Note: GreUnrealizeColorTable is provided for the support of older applications and is of less importance for new applications.

This function reverses GreRealizeColorTable by causing the default physical color table for the device to be reinstated. The logical color table is unaffected by this function.

Support: This function must be supported by the presentation driver. GreUnrealizeColorTable is called by GpiUnRealizeColorTable in response to the request of an application to restore the application's logical color table prior to the last call to GpiRealizeColorTable. This function can be handled by bit-map simulation.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreUnrealizeColorTable

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COL_TABLE_NOT_REALIZED
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Chapter 9. Mandatory Functions for Display Drivers

This chapter describes the functions that must be supported by the display driver for the primary screen. Although these functions are not required for hardcopy drivers, hardcopy drivers should provide a default routine to handle the functions described in this chapter, and at FillLogicalDeviceBlock time, put a pointer to the default routine in the relevant entries in the dispatch table. The default routine will post a warning PMERR_DEV_FUNC_NOT_INSTALLED and return BOOLEAN TRUE.

Descriptions of these mandatory functions for display drivers are provided. The functions are grouped according to the conditional include sections of the header file:

- AVIO functions (INCL_AVIOP, INCL_GRE_DEVMISC1)
- Bit-map functions (INCL_GRE_BITMAPS)
- Device functions 2 (INCL_GRE_DEVMISC2)
- Device functions 3 (INCL_GRE_DEVMISC3)
- Miscellaneous screen functions (INCL_GRE_PICK, INCL_WINPOINTERS).

Each description shows what the handling routine is expected to do, the parameters passed to the routine, and the values that the routine returns.

AVIO Functions

Advanced VIO (AVIO) functions are used to display characters. These functions must be supported for Display DCs. Hardcopy drivers do not support AVIO functions. When writing to an AVIO presentation space, an application must ensure that windows containing alphanumeric data are device-cell aligned, where appropriate. The presentation driver can determine whether any characters, which are not cell aligned, are visible. Most column, row, length, width, and height values correspond to cells within the presentation space Logical Video Buffer (LVB) whose origin is assumed to be at the bottom-left corner of the buffer (0,0).

The presentation driver is expected to clip alphanumeric data to the DC region. This is performed in the same way as for normal graphics, by enumerating the rectangles using GreGetClipRects and clipping each line to a single rectangle. Although the presentation driver is neither expected to test for correlation hits nor to accumulate GPI_BOUNDS, it should accumulate USER_BOUNDS for AVIO functions if the COM_ALT_BOUNDS command flag is set.

The VIO presentation space is passed to the display driver as a pointer to a VioPresentationSpace structure. The display driver uses this structure to extract the current state data to allow it to update the display. The VioPresentationSpace structure is defined as:

PresentationSpaceLock Lock (not used by the presentation driver).

pLVB Pointer to the LVB. The LVB is a two-dimensional array of character cells and is assumed to begin at offset zero within the segment. The presentation driver must not alter the contents of this field.

pBVSCB Not used by the presentation driver.

rgfAVio Not used by the presentation driver.

CellByteSize Size in bytes of a cell in the logical video buffer (LVB). This value is either *2* or *4*. The presentation driver must not alter the contents of this field.

BufferRowCount Number of cell rows in the logical video buffer. The presentation driver must not alter the contents of this field.

BufferColumnCount	Number of cell columns in the logical video buffer. The presentation driver must not alter the contents of this field.
WindowOriginRow	Row index for the logical video buffer. This field, together with the parameter WindowOriginColumn, indicates the logical video buffer cell that is drawn in the bottom left of the window's client area. The presentation driver must not alter the contents of this field.
WindowOriginColumn	Column index for logical video buffer (see WindowOriginRow). The presentation driver must not alter the contents of this field.
TextCursorRow	Row coordinate of flashing text cursor relative to the logical video buffer. The presentation driver must not alter the contents of this field.
TextCursorColumn	Column coordinate of flashing text cursor relative to the logical video buffer. The presentation driver must not alter the contents of this field.
TextCursorStartLine	First scan line of a character-cell image overlaid by the text cursor. Lines in the cell image are numbered from top-to-bottom. The first line is *0*. The presentation driver must not alter the contents of this field.
TextCursorEndLine	Last scan line of a character-cell image, overlaid by the text cursor. The presentation driver must not alter the contents of this field.
TextCursorWidth	Width of text cursor in pels. The presentation driver must not alter the contents of this field.
TextCursorVisible	Indicates whether the cursor is visible (non-zero) or invisible (zero). The presentation driver must not alter the contents of this field.
CellImageHeight	Height of character cell in pels. When the value passed is *0* or invalid, the presentation driver should reset it to the device default value.
CellImageWidth	Width of character cell in pels. When the value passed is *0* or invalid, the presentation driver should reset it to the device default value.
CodePageID	ID of current code page for this presentation space. When the value passed is *0* or invalid, the presentation driver should reset it to the device default value.
WindowHeight	Not used by the presentation driver.
WindowWidth	Not used by the presentation driver.
hConsoleDisplayContext	Device context handle associated with the presentation space. The presentation driver must not alter the contents of this field.
hVioWindow	Not used by the display driver.
RowOrgLatch	See ColOrgLatch.
ColOrgLatch	Window origin coordinates, WindowOriginRow and WindowOriginColumn, saved on completion of the last call to GreUpdateCursor. RowOrgLatch and ColOrgLatch are used by the display driver to record the state of the currently displayed cursor. They are interrogated by GreUpdateCursor to determine whether the cursor has moved.
CursorRow	See CursorCol.
CursorCol	Cursor coordinates, TextCursorRow and TextCursorColumn, saved on completion of the last call to GreUpdateCursor. These fields are used by the display driver to record the state of the currently displayed cursor so that it can be successfully erased.
CursorStartLine	See CursorEndLine.

CursorEndLine Cursor start and end lines, TextCursorStartLine and TextCursorEndLine, saved on completion of the last call to GreUpdateCursor. These fields are used by the display driver to record the state of the currently displayed cursor so that it can be successfully erased.

CursorWidth Cursor width (TextCursorWidth) saved on completion of the last call to GreUpdateCursor. This field is used by the display driver to record the state of the currently displayed cursor so that it can be successfully erased.

PartialCellAdjust This is a negative value reflecting the partial cell height that must be below the bottom of the window to ensure that a complete cell is positioned at the top of the window (actual window height = WindowHeight rounded up to next complete character cell − PartialCellAdjust). The presentation driver must not alter the contents of this field.

XLatch See YLatch.

YLatch Pel coordinates of the bottom-left corner of the cursor rectangle relative to the bottom-left corner of the window.

WidthLatch See HeightLatch.

HeightLatch The height and width in pels of the cursor saved on completion of the last call to GreUpdateCursor. These are taken to be CellImageHeight and the difference between TextCursorStartLine and TextCursorEndLine, taking account of any wrapping. The width and height latches are used by GreUpdateCursor to record the screen region corresponding to an exclusive-OR cursor.

CellHeightLatch The height in pels of the character cell for which the cursor was last drawn. This parameter is used to detect cell height changes. These values were saved on completion of the last call to GreUpdateCursor.

rgfShieldStates Flags:

- **CursorShowing (0x0001)** Cursor is visible on the screen. This flag is maintained by display driver.
- **fHasTheFocus (0x0002)** This window has the input focus. This flag is not modified by display driver.
- **fCursorIsOn (0x0004)** Set to indicate that this is the *on* phase of the blink cycle. The cursor should be invisible during the *off* phase of the blink cycle. This flag is not modified by display driver.

pFontsLoaded[3] Array of pointers for AVIO fonts 1, 2, and 3

pMapFontsLoaded[3] Array of pointers to code page maps for AVIO Fonts 1, 2, and 3.

FormatID Presentation-space format. This identifies the format of the attribute bytes in the LVB:

0H CGA
0H Extended
70H World-wide format.

The presentation driver must not alter the value of this field.

lpNLSExt Pointer to an National Language Support (NLS) extension structure where Double Byte Character Set (DBCS) environment vectors are set for AVIO Fonts 1, 2, and 3. If the display driver is required to display DBCS, it must maintain the array of DBCSEvInfox4x, instead of DBCSEvInfox0x. When the display driver supports NLS, it must set the DBCS vectors for lcid 1−3.

Each character cell is contained in a 2-byte or 4-byte array in the LVB. The format of the character-cell array is:

Code Point Position of the character in the code table

CGA Attribute Byte Character attributes. The four low-order bits represent the foreground color and the high-order bits represent the character background color. Each 4-bit color value corresponds to an explicit 24-bit RGB value. The RGB values are defined within the graphics engine and match the colors available on a CGA device.

Extended Attribute Byte Applies only to 4-byte cells. It is defined as:

- **Bit 7** Underscore
- **Bit 6** Reverse video
- **Bit 4** Background transparency. When set, the background is transparent. When clear, the background is opaque.
- **Bits 1–0** Character Set 0, 1, 2, or 3.

Extended Attribute Byte *(World-wide format)*. Applies only to 4-byte cells. It is defined as:

- **Bit 7** Underscore
- **Bit 6** Reverse video
- **Bit 4** Background transparency. When set, the background is transparent. When clear, the background is opaque.
- **Bit 3** Left vertical grid
- **Bit 4** Top horizontal grid
- **Bits 1–0** Character Set 0, 1, 2, or 3.

Spare Attribute Byte Applies only to 4-byte character cells. It is reserved for the system.

Spare Attribute Byte *(World-wide format)*. Applies only to 4-byte character cells.

- **Bit 7** DBCS trailing byte maintained by the operating system
- **Bits 6–1** Reserved for the system
- **Bit 0** DBCS byte maintained by the operating system.

Notice that applications are not allowed to set bit 7 and bit 0.

Mandatory Functions (for Display Drivers) by Category

Related mandatory functions for display drivers can be grouped together into the following categories:

AVIO Functions

- GreCharRect (see page 9-6)
- GreCharStr (see page 9-7)
- GreDeviceSetAVIOFont (see page 9-10)
- GreScrollRect (see page 9-18)
- GreUpdateCursor (see page 9-22)

Bit-Map Functions

- GreDeviceSetCursor (see page 9-11)
- GreRestoreScreenBits (see page 9-14)
- GreSaveScreenBits (see page 9-17)

Device Functions 2

- GreDeviceInvalidateVisRegion (see page 9-9)
- GreGetStyleRatio (see page 9-13)
- GreSetStyleRatio (see page 9-21)

Device Functions 3

- GreDeath (see page 9-8)
- GreResurrection (see page 9-16)

Miscellaneous Screen Functions

- GreGetPickWindow (see page 9-12)
- GreSetColorCursor (see page 9-19)
- GreSetPickWindow (see page 9-20)

GreCharRect

```
#define INCL_AVIOP

LONG GreCharRect (hdc, pVioPS, pCharRect, pInstance, lFunction)
```

This function draws a rectangle of character cells from the LVB to the device context. The attributes for each character are applied by the handling routine as the character is drawn.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pVioPS	VioPresentationSpace *	Pointer to the Vio presentation space.
pCharRect	LPGridRectRef	Pointer to a block of parameters for the call. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCharRect.

pCharRect Pointer to a parameter block. This is a GridRectRef structure:

- **StartRow** The starting row (relative to the bottom left of the LVB) of the character rectangle to be drawn.
- **StartCol** The starting column (relative to the bottom left of the LVB) of the character rectangle to be drawn.
- **RectWidth** The width in character cells of the rectangle to be updated.
- **RectHeight** The height of the rectangle to be updated.

Return Codes: This function returns a LONG value as an error indicator:

NO_ERROR	Successful
CE_INVALID_PRESENTATION_SPACE	Error. For example, invalid CellByteSize.

Remarks: This function is used to implement the advanced Vio function, VioSetOrg.

GreCharStr

```
#define INCL_AVIOP

LONG GreCharStr (hdc, pVioPS, pCharStr, pInstance, lFunction)
```

This function draws a string of character cells from the LVB to the device context. The attributes for each character are applied by the handling routine as the character is drawn. When the end of a row is reached, the next character is drawn in the first cell of the next row. Character drawing continues until either the string or the logical video buffer is exhausted.

Support: This function must be supported by presentation drivers for display devices.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pVioPS	VioPresentationSpace *	Pointer to the Vio presentation space.
pCharStr	LPGridRectRef	Pointer to a block of parameters for the call. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCharStr.

pCharStr Pointer to parameter block. This is a GridStringRef structure:

- **StartRow** The starting row (relative to the bottom left of the LVB) of the character rectangle to be drawn.
- **StartCol** The starting column (relative to the bottom left of the LVB) of the character rectangle to be drawn.
- **StringLength** Number of characters to be written.

Return Codes: This function returns a LONG value as an error indicator:

NO_ERROR	Successful.
GRE_INVALID_COLUMN_INDEX	Invalid column index.
CE_INVALID_PRESENTATION_SPACE	Error. For example, invalid CellByteSize.
CE_INVALID_ROW_INDEX	Invalid row index.
GRE_NEGATIVE_LENGTH	Negative length.

GreDeath

```
#define INCL_GRE_DEVMISC3

BOOL GreDeath (hdc, pInstance, lFunction)
```

This function informs the presentation driver that the entire screen is required by another screen group (an application that is not running under the Presentation Manager interface). Any current Presentation Manager applications are set to the background. While this condition exists, the presentation driver must handle all calls as usual. However, it may not affect the underlying hardware, that is, the presentation driver must continue to accumulate bounds and respond to queries but it may not actually draw to the display. When GreResurrection is called, the missing output will be recreated by the system sending a WM_PAINT message to the application.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeath

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. An error code for conditions that the handling routine is expected to check:

PMERR_DEV_FUNC_NOT_INSTALLED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function goes directly to the Presentation Driver Interface (PDI).

GreDeviceInvalidateVisRegion

```
#define INCL_GRE_DEVMISC2

BOOL GreDeviceInvalidateVisRegion (hdc, cArray, paBlock, pInstance, lFunction)
```

This function notifies the presentation driver that the visible region and DC region of one or more DCs has changed, and that the affected DCs must revalidate their visible regions before drawing in them. The array identified by paBlock contains a series of structures, each of which identifies a DC and supplies the pointer (pInstance) to its instance data. The display driver responds by setting a flag (HDC_IS_DIRTY) in the instance data of each DC identified in array. The handling routines for all drawing functions should check the HDC_IS_DIRTY flag before drawing. If the flag is set, VisRegionNotify (see page 12-6) must be called to revalidate the DC's visible region.

This function allows the system to defer the calculations caused by visible region changes. This enables menus and dialogs to perform more efficiently.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cArray	LONG	Number of elements in the array.
paBlock	PDC_BLOCK	Pointer to a parameter array. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceInvalidateVisRegion.

paBlock Pointer to an array of DC_BLOCK structures:

hdc Device context handle
pDcl Pointer to instance data.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. An error code for conditions that the handling routine is expected to check:

PMERR_DEV_FUNC_NOT_INSTALLED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeviceSetAVIOFont

```
#define INCL_GRE_DEVMISC1

BOOL GreDeviceSetAVIOFont (hdc, pLogFont, pFontDef, lcid, pInstance, lFunction)
```

This function loads or deletes an image font used by the AVIO presentation space associated with the device context. When loaded, the font is used by subsequent GreCharRect, GreCharStr, and GreScrollRect calls to draw the character images for the appropriate AVIO set. lLcid identifiers LCID_AVIO_1, LCID_AVIO_2, and LCID_AVIO_3 correspond to AVIO sets 1, 2, and 3 respectively.

If the font is not acceptable for use with an AVIO presentation space, the handling routine returns FALSE to indicate an error. The font must be a fixed-pitch image (raster) font that matches one of the cell sizes for the default font. If the font does not match a supported cell size, the characters are displayed in the presentation space as black cells.

A possible approach is for the presentation driver to realize the AVIO fonts required and store the pointers to the fonts in the DC instance data structure.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pLogFont	PFATTRS	Pointer to a logical font
pFontDef	PBYTE	Pointer to a physical font data structure
lcid	LONG	Local identifier value of –2, –3, or –4
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceSetAVIOFont

pFontDef A pointer to a physical font data structure. When the value passed is *0*, the handling routine must delete the loaded font corresponding to lLcid. If the high-order bit of the high-order WORD in lLcid is set, the handle is for a device font.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful. The font is acceptable for use with an AVIO presentation space.
FALSE Error.

GreDeviceSetCursor

```
#define INCL_GRE_BITMAPS

BOOL GreDeviceSetCursor (hdc, pptlHotspot, hbm, pInstance, lFunction)
```

This function sets the cursor bit map that defines the cursor shape. GreDeviceSetCursor is subject to all clipping.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pptlHotspot	PPOINTL	Pointer to hot spot coordinates. See below.
hbm	ULONG	Bit-map handle used for the cursor image.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceSetCursor.

pptlHotspot POINTS structure:

- **x** X-position of the hotspot within the cursor bit map
- **y** Y-position of the hotspot within the cursor bit map.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COORDINATE
PMERR_INV_CURSOR_BITMAP
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The handling routine takes the previous cursor bit map and replaces it with the one indicated by hbm. If hbm is NULL, the cursor has no shape and its image is removed from the display screen.

GreGetPickWindow

```
#define INCL_GRE_PICK

BOOL GreGetPickWindow (hdc, pPick, pInstance, lFunction)
```

This function stores (at the location addressed by pPick) a RECTL structure giving the position and size of the pick window in page-coordinate space.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pPick	PRECTL	Pointer to pick window. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetPickWindow.

pPick The pick window is defined as a RECTL structure in page coordinate space:

xLeft Minimum x-coordinate of window
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of window
yTop Maximum y-coordinate.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetStyleRatio

```
#define INCL_GRE_DEVMISC2

BOOL GreGetStyleRatio (hdc, pRatio, pInstance, lFunction)
```

This function stores the style ratio x-direction and y-direction step values at the location addressed by pRatio. When the line type is LINETYPE_ALTERNATE, the handling routine must restore a value of *0* for the style ratio to ensure that the style mask is rotated after each pel is drawn. See "Line Attributes" on page 8-3.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pRatio	PBYTE	Pointer to style-ratio value. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetStyleRatio.

pRatio The style ratio is defined as a two-byte value. The low-order byte indicates a step in the x-direction, the high-order byte a step in the y-direction.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreRestoreScreenBits

```
#define INCL_GRE_BITMAPS

BOOL GreRestoreScreenBits (hdc, hsbBits, prclRect, flOptions, pInstance, lFunction)
```

This function restores a rectangle of bits to a screen rectangle and can also free the handle of the saved bits.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hsbBits	ULONG	Handle to screen bits to be restored.
prclRect	PRECTL	Pointer to a screen rectangle defined in screen coordinates.
flOptions	ULONG	Options flags. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRestoreScreenBits.

flOptions Option flags, valid values are:

RSB_FREE *1* (free the save bits handle)
RSB_RESTORE *2* (restore the bits to the screen)

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INCOMPATIBLE_BITMAP
PMERR_INCORRECT_DC_TYPE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_BITBLT_MIX
PMERR_INV_BITBLT_STYLE
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR

PMERR_INV_CHAR_SET_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_INDEX
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_DC_DATA
PMERR_INV_DC_TYPE
PMERR_INV_DRIVER_NAME
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_END_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SCAN_START
PMERR_INV_SETID
PMERR_INV_USAGE_PARM
PMERR_REALIZE_NOT_SUPPORTED
PMERR_UNSUPPORTED_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: Clipping is done on the restored bits, as necessary.

GreResurrection

```
#define INCL_GRE_DEVMISC3

LONG GreResurection (hdc, cbVmem, pReserved, pInstance, lFunction)
```

This function reverses the condition set by GreDeath and restores the screen to the Presentation Manager interface. Presentation Manager applications are set to the foreground. The presentation driver is enabled to update the screen for subsequent drawing calls.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cbVmem	LONG	Number video memory bytes changed. See below.
pReserved	PULONG	Reserved pointer. Must be *0*.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreResurrection.

cbVmem The number of bytes of video memory that have been corrupted (determined by the VIO). The display driver can use this value to determine whether any of its video memory has been destroyed by the application. Some display drivers can ignore this parameter.

Return Codes: On completion, the handling routine must return a LONG value (lResult):

0 Error
1 The screen has been successfully redrawn.
2 The screen has not been completely redrawn, further action is required from the application.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function goes directly to the Presentation Driver Interface (PDI).

GreSaveScreenBits

```
#define INCL_GRE_BITMAPS

ULONG GreSaveScreenBits (hdc, prclRect, pInstance, lFunction)
```

This function saves a rectangle of screen bits.

Support: This function must be supported by display drivers. It is permissible to implement this function by returning *0* to indicate that the bits were not saved, and therefore, must be saved by the calling routine.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
prclRect	PRECTL	Pointer to a screen rectangle defined in screen coordinates
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSaveScreenBits

Return Codes: On completion, the handling routine must return a handle to the saved bits (hsbBits) or *0* to indicate that the bits were not saved or that an error occurred.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function lets the user-interface routines improve the performance of dialog boxes.

GreScrollRect

```
#define INCL_AVIOP

LONG GreScrollRect (hdc, pVioPS, paScrollRect, pInstance, lFunction)
```

This function scrolls the contents of the LVB through the DC. The contents of the LVB are not affected by this function. Typically, the presentation driver responds to this call by calling GreCharRect. An alternative approach is to use the horizontal and vertical movement fields to define a new source rectangle in the DC and to use GreBitblt to transfer the bits. When new information is revealed from the LVB as a result of the scroll, the handling routine calls GreCharRect to update the display. This approach provides considerable performance advantages for devices that support GreBitblt. See "GreBitblt" on page 8-26.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pVioPS	VioPresentationSpace *	Pointer to the Vio presentation space.
paScrollRect	LPScrollRectRef	Pointer to a block of parameters for the call. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreScrollRect.

paScrollRect Pointer to parameter block for this function. This block is defined as a ScrollRectRef structure:

- **StartRow** Starting row relative to the bottom left of the LVB
- **StartCol** Starting column in the logical video buffer of the character string to be output
- **RectWidth** Width of the scroll rectangle
- **RectDepth** Depth of the scroll rectangle
- **HorzMovement** Number of columns to be scrolled
- **VertMovement** Number of rows to be scrolled

  Note: Positive values mean movement downward or to the right, negative mean upward or to the left.
- **lpFillCell** Pointer to a cell containing the character and attributes to be used for filling the tail of the scroll region. This pointer is only used when GreBitblt is used to implement this function. When this lpFillCell is passed as NULL, the logical video buffer has been updated. The handling routine then must call GreCharRect to update the display.

Return Codes: This function returns a LONG value as an error indicator:

NO_ERROR	Successful
CE_INVALID_PRESENTATION_SPACE	Error. For example, invalid CellByteSize

GreSetColorCursor

```
#define INCL_WINPOINTERS

BOOL GreSetColorCursor (hdc, pPointerInfo, pInstance, lFunction)
```

This function sets the bit maps that define a color cursor or pointer. The handling routine in the presentation driver updates its copy of the pointer definition to that identified by pPointerInfo.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pPointerInfo	PPOINTERINFO	Pointer to pointer information. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceSetCursor.

pPointerInfo Pointer to a POINTERINFO structure. This structure is described in the *OS/2 2.0 Presentation Manager Programming Reference.*

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_COORDINATE
PMERR_INV_CURSOR_BITMAP
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetPickWindow

```
#define INCL_GRE_PICK

BOOL GreSetPickWindow (hdc, pPick, pInstance, lFunction)
```

This function sets the position and size of the pick window in page-coordinate space for subsequent correlation operations.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pPick	PRECTL	Pointer to pick window. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetPickWindow.

pPick The pick window is defined as a RECTL structure in page-coordinate space:

xLeft Minimum x-coordinate of window
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of window
yTop Maximum y-coordinate.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The boundary of the pick window is included in the correlated area.

GreSetStyleRatio

```
#define INCL_GRE_DEVMISC2

BOOL GreSetStyleRatio (hdc, pRatio, pInstance, lFunction)
```

This function sets the style ratio used by the presentation driver's line-drawing algorithm to determine which pels should be set *on* for a sloping line. Display drivers must support this function so that a hardcopy driver (whose device can have a different style ratio) can use the display driver to draw into a bit map that the hardcopy driver can use.

Support: This function must be supported by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pRatio	PBYTE	Pointer to two unsigned bytes corresponding to the aspect of the pels on which a line is drawn
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetStyleRatio

pRatio The style ratio is defined as a two-byte value. The low-order byte indicates a step in the x-direction, and the high-order byte a step in the y-direction. Typical values for style ratios are:

- For EGA devices, x-direction equals 64 and y-direction equals 85
- For one-to-one devices, x-direction equals 64 and y-direction equals 64. In this case, the style ratio steps are set to 64:64 rather than 1:1 to ensure that a single dot in a line-style pattern is a sensible length. The length of a single dot in the pattern is (256/step_value) pels.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreUpdateCursor

```
#define INCL_AVIOP

LONG GreUpdateCursor (hdc, pVioPS, pInstance, lFunction)
```

This function updates the drawn alphanumeric cursor to match the cursor state information contained in the presentation space. This usually involves removing the previous cursor from the window and drawing the new cursor, if visible, according to the presentation space information. The new cursor (if visible) is positioned and clipped according to this information and the window's cell-buffer origin and size.

The cursor is drawn as an exclusive-OR bar. Its new position, size and shape are saved by the handling routine in the Vio presentation space. Only one cursor can be visible on the screen at any time and this must be in the window with the input focus. This is enforced by the operating system for VIO functions but not for AVIO. The AVIO application must alter the visibility of the cursor when changing input focus. When the text cursor collides with an AVIO and VIO drawing, the presentation driver must remove and redraw the cursor around the alphanumeric updates.

Note: GreBitblt copies everything including the cursor.

The presentation driver can assume that the values in the RowOrgLatch and CursorWidth fields of the VioPresentationSpace structure parallel the WindowOriginRow and TextCursorWidth, respectively.

Support: This function must be supported by display drivers.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pVioPS	VioPresentationSpace *	Pointer to the Vio presentation space
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreUpdateCursor

Return Codes: This function returns a LONG value as an error indicator:

NO_ERROR Successful
CE_INVALID_PRESENTATION_SPACE Error. For example, invalid CellByteSize.

Chapter 10. Simulated Functions

These functions are simulated by handling routines in the graphics engine and are called through pointers in the default dispatch table. Any simulated functions can be hooked to improve performance or to exploit special features of the device. *Hooking* is performed by overwriting the pointer in the presentation driver's copy of the dispatch table with a pointer to the presentation driver's handling routine. If this is done, the the original pointer *must* be saved in order to pass calls to the engine's handling routine.

When the presentation driver has hooked a function, all calls to that function are passed through the dispatch table directly to the driver's handling routine. If the presentation driver cannot completely handle a hooked function, it can pass the call to the engine's routine for completion.

The functions in this chapter are grouped according to the conditional include sections of the header file:

- Arc functions (INCL_GRE_ARCS)
- Area and Path functions (INCL_GRE_PATHS)
- Clip functions (INCL_GRE_CLIP)
- Line functions (INCL_GRE_LINE)
- Palette Manager functions (INCL_GRE_PALETTE)
- Region functions (INCL_GRE_REGIONS)
- Transform functions (INCL_GRE_XFORMS)

Each description shows what the handling routine is expected to do, the parameters passed to the routine, and the values that the routine returns.

Arc Functions

Drawing functions such as those listed above pass individual drawing orders to the graphics engine. The graphics engine then draws, correlates and takes, or takes bounds on the drawing primitives as directed by the flags. The graphics engine is assumed to clip to the appropriate part of the window, which is the region excluding any window border or frills.

Coordinates are passed as signed 32-bit numbers in a logical space called *world-coordinate space. Angles* are also passed as signed 32-bit numbers. Zero refers to the direction of the positive x-axis, 2^{31} represents 360°. Positive values are counterclockwise from the positive x-axis.

Area and Path Functions

A *path* is an area, or a shape, that can be used to define:

- Wide lines and curves to which changes of scale can be applied
- Shapes and areas for filling
- Irregular shapes to which subsequent primitives are clipped. This is known as a *clip path.*

Clip Functions

Clip regions are defined as rectangles in world coordinates. The boundaries of a clip region rectangle are inclusive of the rectangle they define.

Region Functions

Regions are defined as rectangles in device coordinates. They are inclusive at the bottom-left boundary and exclusive at the top-right boundary. That is, the top-right coordinates are outside the rectangle they define and the bottom-left coordinates are inside the rectangle. When both coordinate pairs are equal, the rectangle dimension is *0*.

Transform Functions

Transform functions provide a complete viewing pipeline whereby coordinates are transformed from world-coordinates to model space to presentation page to device space. For more information, refer to the *OS/2 2.0 Programming Guide Volume III — Graphics Programming Interface*. Figure 10-1 on page 10-3 diagrams the viewing pipeline.

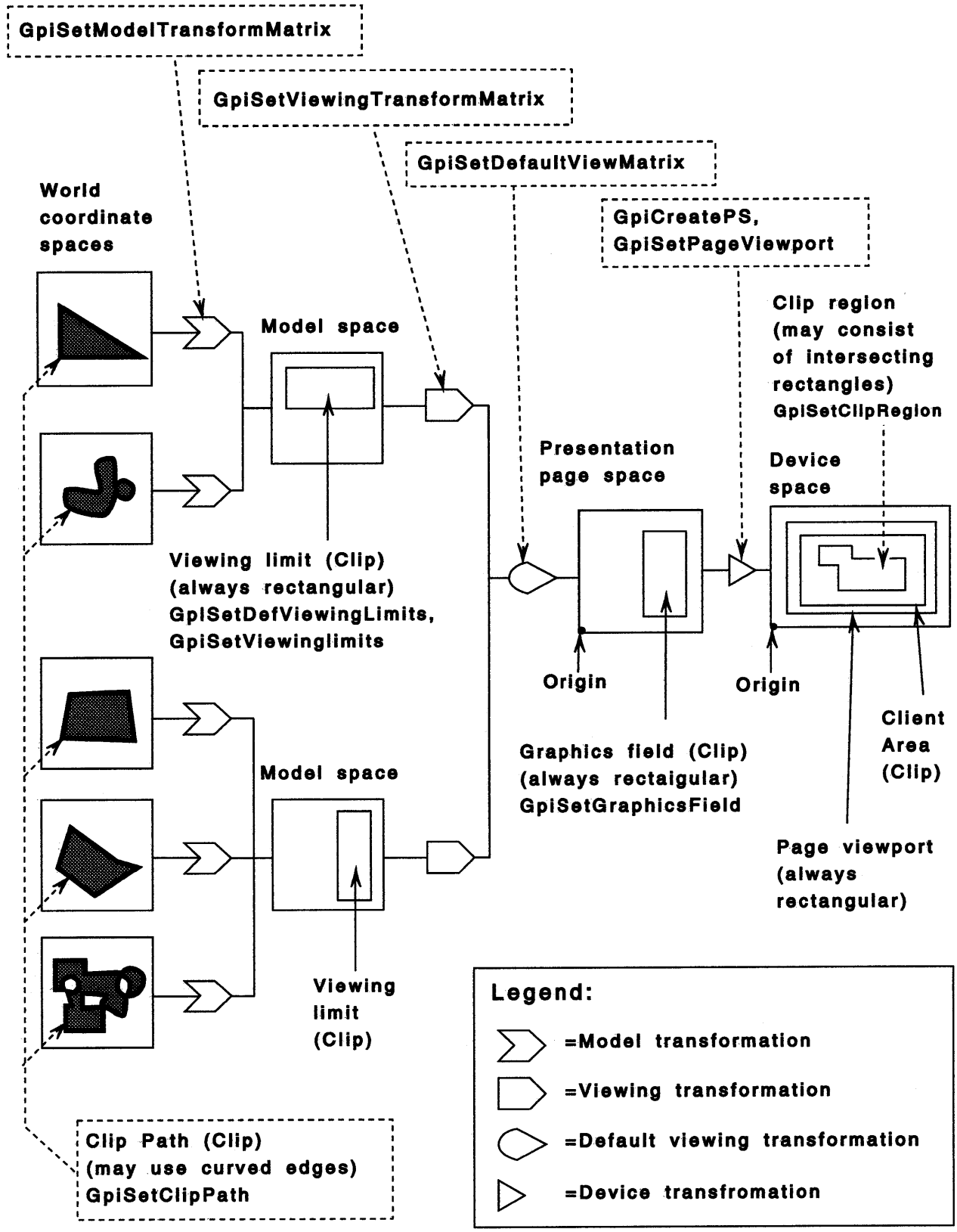


Figure 10-1. Transform and Clipping Pipeline.

Matrix Element Format

The matrix elements for the model and viewing transforms are held in XFORM structures:

fxM11
fxM12
fxM21
fxM22
lM41
lM42

M11, M12, M21, and M22 are fixed-point numbers represented as signed 4-byte integers with a notional binary point between bits 16 and 15:

```
+2.5 is represented by 00028000H
-2.5 is represented by FFFD8000H
-0.5 is represented by FFFF8000H
```

M41 and M42 are signed 4-byte numbers.

Device Transform Definition by Presentation Page Viewport

For a transform, defined by viewport and window rectangles whose bottom-left and top-right coordinates are represented by (X1, Y1), (X2, Y2), (X3, Y3), and (X4, Y4), respectively, the matrix elements are determined as shown below. The point (X3-1/2, Y3-1/2) transforms to (X1-1/2, Y1-1/2), and the point (X4+1/2, Y4+1/2) transforms to (X2+1/2, Y2+1/2). Therefore:

```
M12 = 0
M21 = 0

If X4 >= X3 then
M11 = (X2-X1+1) / (X4-X3+1)
M41 = (X1*X4-X3*X2+1/2 * (X2-X4 + X1-X3)) / (X4-X3+1)

If X4 < X3 then
M11 = (X2-X1+1) / (X4-X3-1)
M41 = (X1*X4-X3*X2-1/2 * (X2+X4+X1+X3)) / (X4-X3-1)

If Y4 >= Y3 then
M22 = (Y2-Y1+1) / (Y4-Y3+1)
M42 = (Y1*Y4-Y3*Y2+1/2 * (Y2-Y4 + Y1-Y3)) / (Y4-Y3+1)

If Y4 < Y3 then
M22 = (Y2-Y1+1) / (Y4-Y3-1)
M42 = (Y1*Y4-Y3*Y2-1/2 * (Y2+Y4+Y1+Y3)) / (Y4-Y3-1)
```

Note: X4 is always greater than X3 and Y4 is always greater than Y3.

In the case of device transforms, (X3, Y3) is always (0, 0), Y4 is always greater than Y3, and the device space coordinates (X2, Y2) are exclusive. This simplifies the formula to:

```
M12 = 0
M21 = 0

M11 = (X2-X1) / (X4+1)
M41 = (X1*X4+1/2 * (X2-1-X4+X1)) / (X4+1) = X1+1/2 (M11-1)

M22 = (Y2-Y1) / (Y4+1)
M42 = (Y1*Y4+1/2 * (Y2-1-Y4+Y1)) / (Y4+1) = Y1+1/2 (M22-1)
```

Bounds, Correlation, and Clipping

All presentation drivers must support bounds computation for both GPI bounds (COM_BOUND) and user bounds (COM_ALT_BOUND). Bounds are calculated on unclipped primitives for all operations that draw to the device, including AVIO functions. GPI bounds are passed in model space coordinates. User bounds are calculated in device-coordinate space. To prevent inaccuracies from occurring when the transform changes, the typical presentation driver maintains bounds in both coordinate sets in its instance data structure. It then accumulates the transforms as they occur.

Correlation is performed in page-coordinate space on the output of primitives that have been clipped only to the viewing limits and graphics field. Correlation is also performed on all operations that draw to the device, except the AVIO function. Notice that hardcopy drivers are not required to perform correlation.

Simulated Functions by Category

Related simulated functions can be grouped together into the following categories:

Arc Functions

- GreArc (see page 10-8)
- GreBoxBoth (see page 10-15)
- GreBoxBoundary (see page 10-17)
- GreBoxInterior (see page 10-19)
- GreFullArcBoth (see page 10-49)
- GreFullArcBoundary (see page 10-51)
- GreFullArcInterior (see page 10-53)
- GreGetArcParameters (see page 10-55)
- GrePartialArc (see page 10-78)
- GrePolyFillet (see page 10-80)
- GrePolyFilletSharp (see page 10-82)
- GrePolySpline (see page 10-84)
- GreSetArcParameters (see page 10-111)

Area and Path Functions

- GreAreaSetAttributes (see page 10-10)
- GreBeginArea (see page 10-11)
- GreBeginPath (see page 10-13)
- GreCloseFigure (see page 10-21)
- GreEndArea (see page 10-41)
- GreEndPath (see page 10-43)
- GreFillPath (see page 10-47)
- GreModifyPath (see page 10-71)
- GreOutlinePath (see page 10-76)
- GreRestorePath (see page 10-98)
- GreSavePath (see page 10-102)
- GreSelectClipPath (see page 10-106)
- GreStrokePath (see page 10-128)

Clip Functions

- GreCopyClipRegion (see page 10-28)
- GreExcludeClipRectangle (see page 10-45)
- GreGetClipBox (see page 10-56)
- GreGetClipRects (see page 10-57)
- GreIntersectClipRectangle (see page 10-69)
- GreOffsetClipRegion (see page 10-74)
- GrePtVisible (see page 10-89)
- GreQueryClipRegion (see page 10-90)
- GreRectVisible (see page 10-96)
- GreRegionSelectBitmap (see page 10-97)
- GreRestoreRegion (see page 10-99)
- GreSaveRegion (see page 10-103)
- GreSelectClipRegion (see page 10-108)
- GreSelectPathRegion (see page 10-110)
- GreSetupDC (see page 10-126)
- GreSetXformRect (see page 10-125)

Line Functions

- GreDrawRLE (see page 10-39)
- GrePolygonSet (see page 10-86)

Palette Manager Functions

- GreDeviceAnimatePalette (see page 10-32)
- GreDeviceCreatePalette (see page 10-33)
- GreDeviceDeletePalette (see page 10-35)
- GreDeviceResizePalette (see page 10-37)
- GreDeviceSetPaletteEntries (see page 10-38)
- GreQueryHWPaletteInfo (see page 10-91)
- GreQueryPaletteRealization (see page 10-92)
- GreRealizePalette (see page 10-93)
- GreUpdateColors (see page 10-130)

Region Functions

- GreCombineRectRegion (see page 10-22)
- GreCombineRegion (see page 10-23)
- GreCombineShortLineRegion (see page 10-24)
- GreCreateRectRegion (see page 10-30)
- GreDestroyRegion (see page 10-31)
- GreEqualRegion (see page 10-44)
- GreGetRegionBox (see page 10-64)
- GreGetRegionRects (see page 10-65)
- GreOffsetRegion (see page 10-75)
- GrePaintRegion (see page 10-77)
- GrePtInRegion (see page 10-88)
- GreRectInRegion (see page 10-95)
- GreSetRectRegion (see page 10-121)

Transform Functions

GreArc

```
#define INCL_GRE_ARCS

LONG GreArc (hdc, paptlPoint, pInstance, lFunction)
```

This function draws an arc through the three points, which are the current position, and the two points specified in the data structure. Upon completion, the current position is the third point of the arc. If GreArc is used within a path definition or an area definition to continue a figure following a GreBox*xxx* or GreFullArc*xxx* function, the error PMERR_INV_NESTED_FIGURES is posted. This is because the GreBox*xxx* and GreFullArc*xxx* functions generate a closed figure within an area or path definition.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
paptlPoint	PPOINTL	Pointer to ArcData array. See below.
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreArc

paptlPoint Pointer to an array of POINTL structures giving the mid and end points of the arc. If the mid-point is coincident with the start or end point, a straight line is drawn from the start point to the end point. If COM_TRANSFORM is not set, the function expects the array of points to be in screen coordinates.

- **x** X-coordinate of point
- **y** Y-coordinate of point.

Return Codes: On completion, this function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreAreaSetAttributes

```
#define INCL_GRE_PATHS

BOOL GreAreaSetAttributes (hdc, lPrimType, flDefsMask, flAttrsMask, pAttrs, pInstance, lFunction)
```

This function is called by the graphics engine after processing a call to GreSetAttrs received inside an area or path bracket. The handling routine in the graphics engine does nothing. Its purpose is to provide an entry in the dispatch table that can be hooked by presentation drivers.

Support: This function must be hooked by presentation drivers that perform their own area or path simulations.

Stack Frame: The parameters passed to GreAreaSetAttributes are identical to those passed to GreSetAttrs.

Parameter	Data Type	Description
hdc	HDC	Device context handle
lPrimType	LONG	Bundle primitive type. See below.
flDefsMask	ULONG	Flags indicating which attributes are to be set to default
flAttrsMask	ULONG	Flags indicating which attributes are to be modified
pAttrs	PBUNDLE	Pointer to the fixed-format bundle record containing the attribute values to be set. See below.
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreAreaSetAttributes

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE Line attribute bundle
PRIM_CHAR Character attribute bundle
PRIM_MARKER Marker attribute bundle
PRIM_AREA Pattern attribute bundle
PRIM_IMAGE Image attribute bundle.

pAttrs This is a pointer to the fixed-format bundle record containing the attribute values to be set as specified by flAttrsMask. Only the attribute fields corresponding to attribute flags set in flAttrsMask, and *not* set in flDefsMask, contain valid values. This buffer must only be large enough to contain data for the highest offset attribute referenced.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

GreBeginArea

```
#define INCL_GRE_PATHS

BOOL GreBeginArea (hdc, flOptions, pInstance, lFunction)
```

This function indicates the beginning of a set of drawing functions that define the boundary of an area. All of the boundaries of the area are considered to be part of the interior, and are filled. GreBeginArea has no direct effect on current position, although it can be affected by drawing orders within the boundary definition. When GreBox*xxx* or GreFullArc*xxx* functions are used within an area definition, they generate closed figures and must not be used within another figure definition. For more information, see GpiBeginArea in the *OS/2 2.0 Presentation Manager Programming Reference*.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
flOptions	ULONG	Option flags. See below.
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreBeginArea

flOptions These flags designate whether the boundary is drawn and what the drawing mode is:

BA_NOBOUNDARY Do not draw boundary lines.
BA_BOUNDARY Draw boundary lines.
BA_ALTERNATE Alternate mode.
BA_WINDING Winding mode.

The defaults are BA_NOBOUNDARY and BA_ALTERNATE.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Area correlation hits are returned at End Area time. No hits are returned for primitives such as lines and arcs that form part of the area definition.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ALREADY_IN_AREA
PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_AREA_CONTROL
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The following functions are valid when received after GreBeginArea and before GreEndArea:

- GreArc
- GreAreaSetAttributes (valid only for color, mix, and valid line attributes)
- GreBoxBoundary
- GreDeviceSetAttributes (valid only for color, mix, and valid line attributes)
- GreDeviceSetGlobalAttribute (valid only for foreground color and mix)
- GreFullArcBoundary
- GrePartialArc
- GrePolyFillet
- GrePolyFilletSharp
- GrePolyLine
- GrePolySpline
- GreQueryCharStringPos
- GreQueryTextBox
- GreSetArcParameters
- GreSetAttributes (valid only for color, mix, and valid line attributes)
- GreSetCurrentPosition
- GreSetGlobalAttribute (valid only for foreground color and mix)
- GreSetModelXform.

GreBeginPath

```
#define INCL_GRE_PATHS

BOOL GreBeginPath (hdc, idPath, pInstance, lFunction)
```

This function identifies the start of a sequence of figures that define a path. Notice that character attribute setting functions are not allowed within a path definition. When GreBox*xxx* or GreFullArc*xxx* functions are used within a path definition, they generate closed figures and must not be used within another figure definition. For more information, see GpiBeginPath in the *OS/2 2.0 Presentation Manager Programming Reference*.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
idPath	LONG	Path identifier. This value must be *1*.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreBeginPath.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ALREADY_IN_PATH
PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PATH_ID
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The following functions are valid when received after GreBeginPath and before GreEndPath:

- GreArc
- GreAreaSetAttributes (valid only for color, mix, and valid line attributes)
- GreBoxBoundary
- GreCharString (outline characters only)
- GreCharStringPos (outline characters only)
- GreCloseFigure
- GreDeviceSetGlobalAttribute (valid only for foreground color and mix)
- GreFullArcBoundary
- GrePartialArc
- GrePolyFillet
- GrePolyFilletSharp
- GrePolyLine
- GrePolyMarker (outline markers only)
- GrePolySpline
- GreQueryCharPositions
- GreQueryTextBox
- GreSetArcParameters
- GreSetAttributes (valid only for color, mix, and valid line attributes)
- GreSetGlobalAttribute (valid only for foreground color and mix)
- GreSetCurrentPosition
- GreSetModelXform.

GreBoxBoth

```
#define INCL_GRE_ARCS

LONG GreBoxBoth (hdc, pBox, pInstance, lFunction)
```

This function draws and fills a rectangular box with one corner at the current (x, y) position and the opposite corner at the specified (x, y) position. The current (x, y) position does not change. When this function occurs within an area or path definition, it generates a closed figure. GreBoxBoth must not occur within any other figure definition.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pBox	PPOINTL	Pointer to BOXDATA. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreBoxBoth.

pBox Pointer to a BOXDATA structure:

ptlOpposite POINTL structure defining the opposite corner of the box. If COM_TRANSFORM is not set, the function expects the point to be in screen coordinates.

- **x** X-coordinate of opposite corner
- **y** Y-coordinate of opposite corner.

lHRound Horizontal length of the full axis of an ellipse. This field is used for rounding each corner.

lVRound Vertical length of the full axis of an ellipse. This field is used for rounding each corner.

Return Codes: On completion, this function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ALREADY_IN_AREA
PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY

PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_AREA_CONTROL
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_BOX_ROUNDING_PARM
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_NESTED_FIGURES
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_NOT_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The sides of the box (before transformation) are parallel to the x-axis and y-axis. The corners of the box can be rounded by means of quarter ellipses of the specified diameters. When the value of either diameter is *0*, no rounding occurs. When the value of either diameter exceeds the length of the corresponding side, that length is used as the diameter instead. When the value of the diameters are equal to the value of the sides, the corners are rounded with a quarter circle. If the current position is (x0, y0), the box is drawn from the current position in a counterclockwise direction.

When correlating, the handling routine records a hit when the pick aperture intersects the boundary or interior, or is completely within the interior (even if the mix used for the fill operation is LEAVEALONE).

GreBoxBoundary

```
#define INCL_GRE_ARCS

LONG GreBoxBoundary (hdc, pBox, pInstance, lFunction)
```

This function draws a rectangular box with one corner at the current (x, y) position and the opposite corner at the specified (x, y) position. The current (x, y) position does not change.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pBox	PPOINTL	Pointer to BOXDATA. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreBoxBoundary.

pBox Pointer to a BOXDATA structure:

ptlOpposite POINTL structure defining the opposite corner of the box. If COM_TRANSFORM is not set, the function expects the point to be in screen coordinates.

- **x** X-coordinate of opposite corner
- **y** Y-coordinate of opposite corner.

lHRound Horizontal length of the full axis of an ellipse. This field is used for rounding each corner.

lVRound Vertical length of the full axis of an ellipse. This field is used for rounding each corner.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_BOX_ROUNDING_PARM
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_NESTED_FIGURES
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The sides of the box (before transformation) are parallel to the x-axis and y-axis. The corners of the box can be rounded by means of quarter ellipses of the specified diameters. When the value of either diameter is *0*, no rounding occurs. When the value of either diameter exceeds the length of the corresponding side, that length is used as the diameter instead. When the value of the diameters are equal to the value of the sides, the corners are rounded with a quarter circle. If the current position is (x0, y0), the box is drawn from the current position in a counterclockwise direction.

When correlating, the handling routine records a hit when the pick aperture intersects the boundary.

GreBoxInterior

```
#define INCL_GRE_ARCS

LONG GreBoxInterior (hdc, pBox, pInstance, lFunction)
```

This function draws a rectangular box with one corner at the current (x, y) position and the opposite corner at the specified (x, y) position. The current (x, y) position does not change. When this function occurs within an area or path definition, it generates a closed figure. GreBoxInterior must not occur within any other figure definition.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pBox	PPOINTL	Pointer to BOXDATA. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreBoxInterior.

pBox Pointer to a BOXDATA structure:

ptlOpposite POINTL structure defining the opposite corner of the box. If COM_TRANSFORM is not set, the function expects the point to be in screen coordinates.

- **x** X-coordinate of opposite corner
- **y** Y-coordinate of opposite corner.

lHRound Horizontal length of the full axis of an ellipse. This field is used for rounding each corner.

lVRound Vertical length of the full axis of an ellipse. This field is used for rounding each corner.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ALREADY_IN_AREA
PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY

PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_AREA_CONTROL
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_BOX_ROUNDING_PARM
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_NESTED_FIGURES
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_NOT_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The sides of the box (before transformation) are parallel to the x-axis and y-axis. The corners of the box can be rounded by means of quarter ellipses of the specified diameters. When the value of either diameter is *0*, no rounding occurs. When the value of either diameter exceeds the length of the corresponding side, that length is used as the diameter instead. When the value of the diameters are equal to the value of the sides, the corners are rounded with a quarter circle. If the current position is (x0, y0), the box is drawn from the current position in a counterclockwise direction. This is significant when, for example, the area mode is BA_WINDING.

When correlating, the handling routine records a hit when the pick aperture intersects, or is completely within, the interior (even if the mix used for the fill operation is LEAVEALONE).

GreCloseFigure

```
#define INCL_GRE_PATHS

BOOL GreCloseFigure (hdc, pInstance, lFunction)
```

This function closes a figure within a path definition by drawing a line from the current (x, y) position to the start point of the figure. Upon completion, the current position is the start point of the figure. *Open figures* can be generated by starting a new figure (with a Move function) or by ending the path without first closing the figure. GreCloseFigure is valid outside of a path definition. When this occurs, this function has no effect and the handling routine ignores it. For more information, see GpiCloseFigure in the *OS/2 2.0 Presentation Manager Programming Reference*.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCloseFigure

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCombineRectRegion

```
#define INCL_GRE_REGIONS

LONG GreCombineRectRegion (hdc, hrgnDst, prclRect, hrgnSrc, cmdMode, pInstance, lFunction)
```

This function combines a region with a rectangle to make a new region. If COM_TRANSFORM is not set, the function expects the point to be in device coordinates. The destination region can be the same as the source region. An error is raised if either of the regions specified is currently selected as the clip region. The source and destination regions must be of the same device class.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgnDst	HRGN	Destination region handle.
prclRect	PRECTL	Pointer to rectangle.
hrgnSrc	HRGN	Region handle.
cmdMode	LONG	Method of combination. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCombineRectRegion.

cmdMode Method of combination:

CRGN_OR	Union of hrgnSrc and prclRect.
CRGN_COPY	prclRect only. hrgnSrc is ignored.
CRGN_XOR	Symmetric difference of hrgnSrc and prclRect.
CRGN_AND	Intersection of hrgnSrc and prclRect.
CRGN_DIFF	hrgnSrc and NOT(prclRect).

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the new region:

RGN_ERROR	Error
RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_INV_REGION_MIX_MODE
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCombineRegion

```
#define INCL_GRE_REGIONS

LONG GreCombineRegion (hdc, hrgnDst, hrgnSrc1, hrgnSrc2, cmdMode, pInstance, lFunction)
```

This function combines two regions to make a third. The destination region can be the same as one of the source regions. An error is raised when any one of the specified regions is currently selected as the clip region. All source and target regions must be of the same device class.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgnDst	HRGN	Destination region handle.
hrgnSrc1	HRGN	First region handle.
hrgnSrc2	HRGN	Second region handle.
cmdMode	LONG	Method of combination. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCombineRegion.

cmdMode Method of combination:

CRGN_OR	Union of hrgnSrc1 and hrgnSrc2.
CRGN_COPY	hrgnSrc1 only. hrgnSrc2 is ignored.
CRGN_XOR	Symmetric difference of hrgnSrc1 and hrgnSrc2.
CRGN_AND	Intersection of hrgnSrc1 and hrgnSrc2.
CRGN_DIFF	hrgnSrc1 and NOT(hrgnSrc2).

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the new region:

RGN_ERROR	Error
RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCombineShortLineRegion

```
#define INCL_GRE_REGIONS

BOOL GreCombineShortLineRegion (hdc, hrgn, pScanData, pInstance, lFunction)
```

This function combines an area lying between polyshortline pairs, which is represented by a SCANDATA structure, with a region. pScanData is ORed into the region. This function is used to build regions for path simulation. The function always expects the points in the shortlines to be in device coordinates. An error is raised when the region specified is currently selected as the clip region.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgn	HRGN	Region handle.
pScanData	PSCANDATA	Pointer to a SCANDATA structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCombineShortLineRegion.

pScanData Pointer to a SCANDATA structure:

psiFirstLeft Pointer to the left end of the first polyshortline
psiLastLeft Pointer the left end of the last polyshortline
psiFirstRight Pointer to right edge of first polyshortline
psiLastRight Pointer to right edge of last polyshortline
c Number of scan lines
rclBound RECTL structure defining the bounding rectangle.

Notice that a polyshortline consists of a list of linked SHORTLINE structures:

sih SHORTLINEHEADER structure:

ulStyle Line style
ulFormat Line format
ptlStart (x, y) position of start
ptlStop (x, y) position of end
lxLeft Left edge of bounding rectangle
lxRight Right edge of bounding rectangle
psihNext Pointer to next shortline
psihPrev Pointer to previous shortline.

This structure is a discrete representation of a curve that starts at point (x0, y0) and ends at point (x1, y1). For each of the (y1−y0+1) rows, there is exactly one x value contained in the x-array. The final point in the series is not drawn.

ax Array of x values, as device coordinates.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreConvert

```
#define INCL_GRE_XFORMS

BOOL GreConvert (hdc, lSrc, lDst, paptlPoint, cPoints, pInstance, lFunction)
```

This function converts the specified coordinates from one coordinate space to another by using the current values of the transforms.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
lSrc	LONG	Source-coordinate space. See below.
lDst	LONG	Target-coordinate space. See below.
paptlPoint	PPOINTL	Pointer to array of (x, y) coordinates to transform. The result is also returned to this parameter.
cPoints	LONG	Count of coordinate pairs in the array.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreConvert.

lSrc These values define the source-coordinate space:

CVTC_WORLD	World-coordinate space.
CVTC_MODEL	Model space.
CVTC_DEFAULTPAGE	Default page-coordinate space.
CVTC_PAGE	Page-coordinate space.
CVTC_DEVICE	Device-coordinate space. Screen coordinates are 32-bit signed integers and are used by the presentation driver as screen pel addresses.

lDst Target-coordinate space defined by the same values as lSrc (see above).

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreConvertWithMatrix

```
#define INCL_GRE_XFORMS

BOOL GreConvertWithMatrix (hdc, paptlPoint, cPoints, paXform, pInstance, lFunction)
```

This function converts a series of points by using the matrix indicated. Other current transform matrixes are ignored.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paptlPoint	PPOINTL	Pointer to array of (x, y) coordinates to transform. The result is also returned to this parameter.
cPoints	LONG	Count of coordinate pairs in the array.
paXform	PXFORM	Pointer to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, and M42.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreConvertWithMatrix.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCopyClipRegion

```
#define INCL_GRE_CLIP

LONG GreCopyClipRegion (hdc, hrgn, prclBounds, flOptions, pInstance, lFunction)
```

This function copies the visible region, clip region, or DC region, and returns the complexity and bounds of the resulting region.

Support: This function is supported by the engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hrgn	HRGN	Visible region handle
prclBounds	PRECTL	Bounding rectangle
flOptions	ULONG	Option flags
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCopyClipRegion

prclBounds Pointer to the bounding rectangle of the returned region. The bounding rectangle is returned in the same coordinate system as the region defined by flOptions. This rectangle is inclusive at the bottom and left boundaries, exclusive at the top and right boundaries. When specified as NULL, the bounding rectangle is not returned.

flOptions These flags determine the type of region to be returned in hrgn:

COPYCRGN_ALLINTERSECT The function must return the intersection of all clipping. This value describes the DC region and is expressed in screen coordinates.

COPYCRGN_VISRGN The function must return a copy of the visible region only. This value is returned in screen coordinates.

COPYCRGN_CLIPRGN The function must return a copy of the clip region only. The clip region is expressed in device coordinates.

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT Rectangular region
RGN_COMPLEX Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCreateRectRegion

```
#define INCL_GRE_REGIONS

HRGN GreCreateRectRegion (hdc, paRegion, cRect, pInstance, lFunction)
```

This function creates a region by taking the OR of a series of rectangles. When no rectangles are specified (that is, cRect is *0*), an empty region is created. If COM_TRANSFORM is not set, the function expects the points to be in device coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paRegion	PRECT	Pointer to the region definition, which is an array of rectangle structures. See below.
cRect	LONG	Number of rectangles in the region definition. If this is *0*, an empty region is created.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCreateRectRegion.

paRegion This is a pointer to an array of rectangles, which defines the region. Each rectangle is described by a RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

For each rectangle, xRight must be equal to, or greater than, xLeft. yTop must be equal to, or greater than, yBottom. The bottom and left boundaries of each rectangle are part of the interior of the region; the top and right boundaries are not.

Return Codes: On completion, the handling routine returns the region handle (hrgn), or RGN_ERROR if an error occurred.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDestroyRegion

```
#define INCL_GRE_REGIONS

BOOL GreDestroyRegion (hdc, hrgn, pInstance, lFunction)
```

This function deletes the specified region unless it has been selected as a clipping region. In this case, an error is raised. When a null region is specified, GreDestroyRegion does not delete any region and returns without logging an error.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hrgn	HRGN	Region handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDestroyRegion

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION.

Notice that when an error occurs, the region is not deleted. Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeviceAnimatePalette

```
#define INCL_GRE_PALETTE

DDIENTRY GreDeviceAnimatePalette (hdc, hdevpal, ulFormat, ulStart, cclr, pclr, pInstance, lFunction)
```

This function is the presentation driver version of GreAnimatePalette. It is called for every device that has the palette from GreAnimatePalette selected into it.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hdevpal	ULONG	Device palette handle
ulFormat	ULONG	Specifies the entry format. Must be LCOLF_CONSECRGB.
ulStart	ULONG	Starting index, that is, first palette entry to change
cclr	ULONG	Count of palette entries to change
pclr	PULONG	Pointer to table of new RGB2 palette entries
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceAnimatePalette

Return Codes: On completion, this function returns the following value:

cclr Count of the hardware palette slots changed.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_DC
PMERR_PAL_ERROR.

GreDeviceCreatePalette

```
#define INCL_GRE_PALETTE

DDIENTRY GreDeviceCreatePalette (hdc, ppalinfo, hdevpal, pInstance, lFunction)
```

This function is called by GreSelectPalette. The presentation driver is expected to perform any allocation and data structure initialization needed for a subsequent GreDeviceRealizePalette to succeed.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
ppalinfo	PPALETTEINFO	Pointer to table of palette data from GreCreatePalette. See below.
hdevpal	ULONG	Device palette handle (NULL, if new). See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceCreatePalette.

ppalinfo The structure of palinfo is as follows:

```
struct _PALETTEINFO { /* palinfo */

    ULONG   flCmd;                  /* Options from create                  */
    ULONG   ulFormat;               /* Specifies format of entries at create */
    ULONG   lStart;                 /* Starting index from create           */
    ULONG   clColorData;            /* Number of elements supplied at create */
    RGB2 ; argb[1];             /* Palette entries                  */

    } PALETTEINFO;
```

hdevpal If hdevpal = *0*, ppalinfo points to a palette information structure, which must be set into a new palette whose handle is returned. The new palette is then selected into the DC.

If hdevpal is non-zero, ppalinfo is ignored, the palette is selected into the DC, and hdevpal is a device palette handle previously returned by the presentation driver. In this case, hdevpal is used to bind the palette with the new ddc. This technique is the driver level mechanism for sharing palettes among contexts on the same device.

Return Codes: On completion, this function returns the following value:

hdevpal Handle to the device palette.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ERROR_NEG.

Remarks: If a palette is selected, it should be used by the presentation driver, where possible.

When an application sets a bit map, the bits are specified as indices into the color table supplied with the bit map. The presentation driver allocates its own copy of the bit map. The driver-level pels contain indices to the nearest color in the default physical palette. Notice that this means GreGetBitmapBits does not necessarily return the same bit map that the application set. *Information is potentially lost if the bit map is taken to a less color-capable system and then brought back to the (more capable) system on which it was created.*

Bit maps have an implicit palette based on their color table. An application can select a palette into a memory DC. However, this causes the color table and the bit-map bits to change to the nearest mapping to the new palette. As with UpdateColors on a memory DC, the application must keep a copy of the original bit map if it needs the original bits after selecting a palette on the memory DC. Unlike a shared device such as the display, calls to GreRealizePalette are not performed on memory DCs. The palette takes effect as soon as it is selected.

Palette-using applications must process the new WM_PALETTECHANGED message and not pass this on to DefWndProc. If DefWndProc receives this message for a foreground window, it causes the default system palette to be realized.

GreDeviceDeletePalette

```
#define INCL_GRE_PALETTE

DDIENTRY GreDeviceDeletePalette (hdc, hdevpal, pInstance, lFunction)
```

This function is called by the graphics engine when a palette is selected out of a device context. It informs the device to delete its instance of the palette for the given device context. It is the responsibility of the presentation driver to determine if any other device contexts are using the palette. In this case, the presentation driver does not free the internal data structures needed for palette realization.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hdevpal	ULONG	Handle to the device palette from GreDeviceCreatePalette
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceDeletePalette

Return Codes: On completion, this function returns the following value:

GPI_OK Successful
ERROR_ZERO Error

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ERROR_ZERO
PMERR_INV_PALETTE.

GreDeviceResizePalette

```
#define INCL_GRE_PALETTE

DDIENTRY GreDeviceResizePalette (hdc, hdevpal, ulSize, pInstance, lFunction)
```

This function changes the size of a logical palette. If the size is reduced, the removed entries are deleted. If the size is increased, the new entries are set to Black.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hdevpal	ULONG	Device palette handle
ulSize	ULONG	New palette size
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceResizePalette

Return Codes: On completion, this function returns the following values:

GPI_OK Successful
GPI_ERROR Error

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_PALETTE

GreDeviceSetPaletteEntries

```
#define INCL_GRE_PALETTE

DDIENTRY GreDeviceSetPaletteEntries (hdc, hdevpal, ulFormat, ulStart, cclr, pclr, pInstance, lFunction)
```

This function changes the entries in a palette. These changes do not become apparent until an application calls WinRealizePalette. If an application changes its palette rapidly and makes those changes immediately apparent, it calls GreAnimatePalette.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hdevpal	ULONG	Device palette handle.
ulFormat	ULONG	Specifies the entry format. Must be LCOLF_CONSECRGB.
ulStart	ULONG	Starting index, that is, first palette entry to change.
cclr	ULONG	Count of palette entries to change.
pclr	PULONG	Pointer to table of new RGB2 palette entries.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeviceSetPaletteEntries.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful, if entries set without error.
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_PALETTE.

GreDrawRLE

```
#define INCL_GRE_LINE

ULONG GreDrawRLE (hdc, pRLEHDR, pInstance, lFunction)
```

This function draws a run-length-encoded shape. If the call is passed back to the graphics engine, it calls the GrePolyScanline entry point. GreDrawRLE is called by the area fill code in the graphics engine. These calling functions query the driver with the CAPS_ADDITIONAL_GRAPHICS index to see if the CAPS_CLIP_FILLS bit is set. If it is set, the data delivered to GreDrawRLE has already been clipped. Notice that all coordinates are passed as screen coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pRLEHDR	PRLEHDR	Pointer to run-length-encoding data header. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDrawRLE.

pRLEHDR Pointer an RLEHDR structure:

```
typedef struct {
                  LONG       lType;
                  BRECTL     brectlBounds;
                  PVOID      pRLE;
                } RLEHDR;
```

GreDrawRLE always supports *lType = 0*. In this format, the *pRLE* field in the RLE header points to an array of POINTL structures (see below). The *brectlBounds* rectangle contains the tightest rectangle that fits around the shape.

The RLE array starts with an (x, y) pair, where *y* is the y-coordinate, and *x* is the number of runs on the line. The number of (x, y) pairs follow, where *x* is the left side of the run and *y* is the right side, exclusive. The next (x, y) pair holds the header of the next line. The y-coordinate must increase. If the number of points is *0*, the end of the data has been reached. For example, the following array of eight POINTL structures defines two scan lines of run-length-encoded data:

```
(3,45), (20,25), (42,56), (100,350),  (Three pairs of runs at Scan Line 45)
(2,46), (19,26), (43,56),             (Two pairs of runs at Scan Line 46)
(0,47)                                (End of RLE data indicator)
```

The presentation driver draws lines between the following device-coordinate-space pairs, inclusive, given the data in the above example:

```
(20 ,45) to (24 ,45)
(42 ,45) to (55 ,45)
(100,45) to (349,45)
(19 ,46) to (25 ,46)
(43 ,46) to (55 ,46)
```

Notice that pels are drawn up to, but not including, the right coordinate of the run. That is, runs are filled inclusive or exclusive.

Return Codes: On completion, the handling routine must return an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreEndArea

```
#define INCL_GRE_PATHS

LONG GreEndArea (hdc, flCancel, pInstance, lFunction)
```

This function indicates the end of a set of drawing functions that define the boundary of an area. If the final (x, y) position is not the same as the starting position of the last figure in the area, the handling routine must close the figure by drawing a line from the current position to the start of the final figure. Upon completion, the current (x, y) position is the last (x, y) position specified in the area boundary unless a closure line was drawn. In this case, the start of the last figure in the area definition becomes the current (x, y) position.

The area fill can be built up in memory or on devices that have hardware assist for area fill in the device. For convex figures, there can be a performance gain in simply recording the start and end pel positions across each scan line. Whatever algorithm is used to fill the area, *the interior fill must be identical in each occurrence*. If it is not identical, bit-map operations can fail to join correctly when copied to the screen.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flCancel	ULONG	Cancel. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEndArea.

flCancel The value of this flag specifies whether this call cancels the area definition:

EA_DRAW Draw the area. This is the default.
EA_CANCEL If set, cancel the area definition. Otherwise, draw the area.

Note: When GreBeginArea is not called and EA_CANCEL is set, GreEndArea is valid but has no effect, thus allowing the handling routine to reset an area bracket to a known state when it has no knowledge of the actual current state.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected) on any part of the interior, regardless of mix.
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_AREA_CONTROL
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_NOT_IN_AREA
PMERR_PATH_LIMIT_EXCEEDED
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreEndPath

```
#define INCL_GRE_PATHS

BOOL GreEndPath (hdc, flCancel, pInstance, lFunction)
```

This function identifies the end of a sequence of figures that define a path. This function is valid outside a path definition but has no effect. When this occurs, the handling routine should ignore it.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flCancel	ULONG	Cancel. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEndPath.

flCancel The value of flCancel specifies whether this call cancels the path definition:

EA_DRAW Create the current path
EA_CANCEL Cancel the path definition.

When GreBeginPath is not called and EA_CANCEL is set, GreEndPath is valid but has no effect, thus allowing the handling routine to reset a path bracket to a known state when it has no knowledge of the actual current state.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_END_PATH_OPTIONS
PMERR_INV_HDC
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreEqualRegion

```
#define INCL_GRE_REGIONS

LONG GreEqualRegion (hdc, hrgnSrc1, hrgnSrc2, pInstance, lFunction)
```

This function checks whether two regions, owned by the device identified by hdc, are identical. An error is raised when either region is currently selected as the clip region.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hrgnSrc1	HRGN	First region handle
hrgnSrc2	HRGN	Second region handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreEqualRegion

Return Codes: This function returns an integer (lEquality) indicating whether the regions are equal:

EQRGN_EQUAL Equal
EQRGN_ERROR Error
EQRGN_NOTEQUAL Not equal.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreExcludeClipRectangle

```
#define INCL_GRE_CLIP

LONG GreExcludeClipRectangle (hdc, prclRect, pInstance, lFunction)
```

This function excludes the specified rectangle from the clipping region, that is, the area covered by the rectangle will be outside the resulting clip region. If COM_TRANSFORM is not set, the function expects the rectangle points to be in device coordinates. GreExcludeClipRectangle creates a clip region when none exists. The application is responsible for deleting this clip region when it is finished. Otherwise, it is not deleted until the DC is closed.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclRect	PRECTL	Pointer to rectangle in world or device coordinates. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD=flags; low-order WORD=NGreExcludeClipRectangle.

prclRect RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

All the boundaries of this rectangle are considered to be part of the interior and are clipped accordingly.

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the DC region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT Rectangular region
RGN_COMPLEX Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreFillPath

```
#define INCL_GRE_PATHS

LONG GreFillPath (hdc, idPath, flOptions, pInstance, lFunction)
```

This function fills the interior of the closed figure defined in the path by using the current pattern attributes. Before filling the path, the handling routine must close any open figures in the path definition. On completion, it must delete the path. All of the boundaries of the area are considered to be part of the interior and are filled.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
idPath	LONG	Path ID. Must be *1*.
flOptions	ULONG	Option flags. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreFillPath.

flOptions These flags determine how the path is to be filled:

FPATH_ALTERNATE Fill is performed by using the odd/even (alternate) rule.
FPATH_WINDING Fill is performed by using the winding rule.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_HDC
PMERR_INV_HRGN

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PATH_ID
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreFullArcBoth

```
#define INCL_GRE_ARCS

LONG GreFullArcBoth (hdc, fxMultiplier, pInstance, lFunction)
```

This function draws and fills a full arc centered on the current (x, y) position. The current line attributes apply to the boundary line, and the current pattern attributes to the interior. The dimensions of the arc are defined as a multiplier that is applied to the current arc parameters. GreFullArcBoth does not affect the current position. When the COM_PATH or the COM_AREA flag is set, this function must raise an error.

When correlating, the handling routine must return a hit when the pick aperture intersects the boundary, or is completely within the interior, even when the mix is LEAVEALONE. See "Mix Modes" on page 8-2.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
fxMultiplier	FIXED	Multiplier. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreFullArcBoth.

fxMultiplier The value of this parameter defines the size of the required arc in relation to an arc drawn with the current arc parameters. The multiplier is a fixed-point value. The high-order WORD contains the integer portion; the low-order WORD contains the fractional portion. A value of 64KB gives a multiplier of *1*. The implementation limit of the multiplier is *255*. Notice that this value must not be negative.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ALREADY_IN_AREA
PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_AREA_CONTROL

PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_MULTIPLIER
PMERR_INV_NESTED_FIGURES
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_NOT_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When correlating, the handling routine records a hit when the pick aperture intersects the boundary or interior, or is completely within the interior (even if the mix used for the fill operation is LEAVEALONE).

GreFullArcBoundary

```
#define INCL_GRE_ARCS

LONG GreFullArcBoundary (hdc, fxMultiplier, pInstance, lFunction)
```

This function draws a line by using the current line attributes around the edge of a full arc centered on the current (x, y) position. The dimensions of the arc are defined as a multiplier that is applied to the current arc parameters. GreFullArc*xxx* functions do not affect the current position.

When correlating, the handling routine must return a hit when the pick aperture intersects the boundary. See "Mix Modes" on page 8-2.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
fxMultiplier	FIXED	Multiplier. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreFullArcBoundary.

fxMultiplier The value of this parameter defines the size of the required arc in relation to an arc drawn with the current arc parameters. The multiplier is a fixed-point value. The high-order WORD contains the integer portion; the low-order WORD contains the fractional portion. A value of 64KB gives a multiplier of *1*. The implementation limit of the multiplier is *255*. Notice that this value must not be negative.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT

PMERR_INV_MULTIPLIER
PMERR_INV_NESTED_FIGURES
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When correlating, the handling routine records a hit when the pick aperture intersects the boundary line.

GreFullArcInterior

```
#define INCL_GRE_ARCS

LONG GreFullArcInterior (hdc, fxMultiplier, pInstance, lFunction)
```

This function draws a filled, full arc by using the current pattern attributes with its center at the current (x, y) position. The dimensions of the arc are defined by a multiplier that is applied to the current arc parameters. GreFullArc*xxx* functions do not affect the current position. The arc boundary is not drawn. When the COM_PATH or the COM_AREA flag is set, this function must raise an error.

When correlating, the handling routine must return a hit when the pick aperture intersects the interior, or is completely within the interior, even when the mix is LEAVEALONE. See "Mix Modes" on page 8-2.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
fxMultiplier	FIXED	Multiplier. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreFullArcInterior.

fxMultiplier This parameter defines the size of the required arc in relation to an arc drawn with the current arc parameters. The multiplier is a fixed-point value. The high-order WORD contains the integer portion; the low-order WORD contains the fractional portion. A value of 64KB gives a multiplier of *1*. The implementation limit of the multiplier is *255*. Notice that this value must be positive or *0*.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ALREADY_IN_AREA
PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_AREA_CONTROL

PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_MULTIPLIER
PMERR_INV_NESTED_FIGURES
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_NOT_IN_AREA
PMERR_NOT_IN_PATH
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When correlating, the handling routine records a hit when the pick aperture intersects, or is completely within, the interior (even if the mix used for the fill operation is *transparent*).

GreGetArcParameters

```
#define INCL_GRE_ARCS

BOOL GreGetArcParameters (hdc, pArcParms, pInstance, lFunction)
```

This function stores the current arc parameters in the buffer addressed by pArcParms.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pArcParms	PARCPARAMS	Pointer to ARCPARAMS structure
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetArcParameters

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetClipBox

```
#define INCL_GRE_CLIP

LONG GreGetClipBox (hdc, prclRect, pInstance, lFunction)
```

This function determines the dimensions of the tightest rectangle around the DC region in world coordinates. The *DC region* is defined as the intersection of the visible region, clip region, viewing limits, graphics field, and clip path. The bounding rectangle is inclusive. Points on its boundary are deemed to be inside it. If COM_TRANSFORM is not set, the rectangle points are returned in screen coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclRect	PRECTL	Pointer to rectangle in world or screen coordinates. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetClipBox.

prclRect RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

prclRect is a NULL rectangle when the value of xRight is less than xLeft, and yTop is less than yBottom.

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the resultant *clipping region*, which is defined as the intersection of all clipping (that is, the clip path, viewing limits, graphics field, clip region, and visible region).

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT Rectangular region
RGN_COMPLEX Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetClipRects

```
#define INCL_GRE_CLIP

BOOL GreGetClipRects (hdc, prclBound, pControl, parclRect, pInstance, lFunction)
```

This function fills the buffer indicated by parclRect with a list of (x, y) coordinate pairs specifying the rectangles, which define the DC region and intersect an optional bounding rectangle (prclRect). The DC region is the intersection of the visible region, clip region, viewing limits, graphics field, and clip path.

Clipping, when performed by the graphics engine, carries a heavy processing overhead. For simple clipping, the presentation driver gains a significant performance advantage by calling this function to enumerate the clip region, and clipping the line to each rectangle in turn.

Note: If COM_TRANSFORM is set, the rectangle points are returned in device coordinates. If COM_TRANSFORM is not set, the rectangle points are returned in screen coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclBound	PRECTL	Pointer to bounding rectangle in device or screen coordinates. See below.
pControl	PRGNRECT	Pointer to control structure. See below.
parclRect	PRECTL	Pointer to array of rectangle (RECTL) structures.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetClipRects.

prclBound Pointer to a RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

Only rectangles intersecting this bounding rectangle are returned. When this pointer is NULL, all rectangles in the region are enumerated. When this pointer is not NULL, each rectangle returned is the intersection of the bounding rectangle with a rectangle in the region.

If COM_TRANSFORM is set, the function expects the bounding rectangle to be in device coordinates. If COM_TRANSFORM is not set, the function expects the bounding rectangle to be in screen coordinates.

pControl Pointer to a control structure containing additional parameters:

lStart The rectangle number to start enumerating (*1* indicates start at the beginning). By updating this value, the function can be called repeatedly to allow for more rectangles than can be stored in the receiving buffer.

crc The number of rectangles that can fit into the buffer (at least 1).

crcReturned Number of rectangles that were written into the buffer. If less than lStart, there are no more rectangles to enumerate.

usDirection The direction in which the rectangles are listed:

RECTDIR_LFRT_TOPBOT	Left-to-right, top-to-bottom
RECTDIR_RTLF_TOPBOT	Right-to-left, top-to-bottom
RECTDIR_LFRT_BOTTOP	Left-to-right, bottom-to-top
RECTDIR_RTLF_BOTTOP	Right-to-left, bottom-to-top

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetGlobalViewingXform

```
#define INCL_GRE_XFORMS

BOOL GreGetGlobalViewingXform (hdc, paXformData, pInstance, lFunction)
```

This function queries the global viewing transform matrix. On completion, paXformData points to an array of two-dimensional values that defines the global viewing transform matrix.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
paXformData	PXFORM	Pointer to return data in which the array of 6 matrix elements (M11, M12, M21, M22, M41, and M42) are to be stored
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetGlobalViewingXform

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetGraphicsField

```
#define INCL_GRE_XFORMS

BOOL GreGetGraphicsField (hdc, prclGraphicsField, pInstance, lFunction)
```

This function loads the buffer indicated by prclGraphicsField with the integer values that identify the boundaries of the graphics field.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
prclGraphicsField	PRECTL	Pointer to graphics field. See below.
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetGraphicsField

prclGraphicsField RECTL structure:

xLeft Minimum x-coordinate of graphics field
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of graphics field
yTop Maximum y-coordinate.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetModelXform

```
#define INCL_GRE_XFORMS

BOOL GreGetModelXform (hdc, paXformData, pInstance, lFunction)
```

This function queries the current model transform matrix. On completion, paXformData is an array of two-dimensional values defining the current model transform matrix.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
paXformData	PXFORM	Pointer to return data in which the array of 6 matrix elements (M11, M12, M21, M22, M41 and M42) are to be stored
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetModelXform

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetPageUnits

```
#define INCL_GRE_XFORMS

ULONG GreGetPageUnits (hdc, pExtent, pInstance, lFunction)
```

This function returns the page units for the specified display context. On completion, the structure addressed by pExtent contains the dimensions of the page expressed in the units indicated by the return value of this function.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pExtent	PSIZEL	Pointer to a SIZEL structure where the page dimensions are returned
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetPageUnits

pExtent On completion, the SIZEL structure contains the actual dimensions of the page:

lWidth Page width
lHeight Page height.

Return Codes: This function returns the page units, or GPI_ERROR if an error occurred. The defined page units are described under "GreSetPageUnits" on page 10-117.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetPageViewport

```
#define INCL_GRE_XFORMS

BOOL GreGetPageViewport (hdc, prclViewport, pInstance, lFunction)
```

This function loads the buffer indicated by prclViewport with the page viewport coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclViewport	PRECTL	Pointer to page viewport boundaries. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetPageViewport.

prclViewport This is a RECTL structure in device coordinates:

xLeft Minimum x-coordinate of viewport
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of viewport
yTop Maximum y-coordinate.

Boundaries are inclusive at the bottom-left corner, and exclusive at the top-right corner.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetRegionBox

```
#define INCL_GRE_REGIONS

LONG GreGetRegionBox (hdc, hrgn, prclRect, pInstance, lFunction)
```

This function loads the buffer pointed to by prclRect with the dimensions of the tightest rectangle around the region indicated by hrgn. Notice that the rectangle is always returned in device coordinates. GreGetRegionBox raises an error when hrgn is the handle of the currently selected clip region.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgn	HRGN	Region handle.
prclRect	PRECTL	Pointer to rectangle in device coordinates. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetRegionBox.

prclRect RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT Rectangular region
RGN_COMPLEX Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When the region is empty, the rectangle is returned with the right boundary equal to the left, and the top boundary equal to the bottom. GreGetRegionBox must return an error when the region indicated by hrgn is selected into the DC as a clip region.

GreGetRegionRects

```
#define INCL_GRE_REGIONS

BOOL GreGetRegionRects (hdc, hrgn, prclBoundRect, pControl, parclRect, pInstance, lFunction)
```

This function fills the array pointed to by parclRect with a list of (x, y) coordinate pairs specifying the rectangles that together define the region associated with hrgn. All rectangle coordinates are inclusive at the bottom-left corner, and exclusive at the top-right corner. Notice that the coordinates of the bounding box, and the rectangles returned, will be in device coordinates. GreGetRegionRects raises an error when hrgn is the handle of a currently selected clip region.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgn	HRGN	Region handle.
prclBoundRect	PRECTL	Pointer to bounding rectangle. See below.
pControl	PRGNRECT	Pointer to control structure. See below.
parclRect	PRECTL	Pointer to array of rectangle structures that define the region.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetRegionRects.

prclBoundRect RECTL structure:

xLeft Minimum x-coordinate of window
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of window
yTop Maximum y-coordinate.

Only rectangles intersecting this bounding rectangle are returned. If this pointer is NULL, all rectangles in the region are enumerated. If this pointer is not NULL, only rectangles that intersect the bounding rectangle are returned. Each of these rectangles is the intersection of the bounding rectangle with a rectangle in the region.

pControl This is a pointer to a RGNRECT structure:

lrcStart The rectangle number to start enumerating (*1* indicates start at the beginning).

crc The number of rectangles that can fit into the buffer (at least 1).

crcReturned Number of rectangles that were written into the buffer. If less than lrcStart, there are no more rectangles to enumerate.

usDirection The direction in which the rectangles are listed:

RECTDIR_LFRT_TOPBOT Left-to-right, top-to-bottom
RECTDIR_RTLF_TOPBOT Right-to-left, top-to-bottom
RECTDIR_LFRT_BOTTOP Left-to-right, bottom-to-top
RECTDIR_RTLF_BOTTOP Right-to-left, bottom-to-top.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetViewingLimits

```
#define INCL_GRE_XFORMS

BOOL GreGetViewingLimits (hdc, prclViewingLimits, pInstance, lFunction)
```

This function loads the array indicated by prclViewingLimits with the boundaries of the viewing window in graphic model-space coordinates. All boundaries are inclusive.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclViewingLimits	PRECTL	Pointer to limits of viewing area. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetViewingLimits.

prclViewingLimits Pointer to a RECTL structure:

xLeft	Minimum x-coordinate of viewing limits
yBottom	Minimum y-coordinate
xRight	Maximum x-coordinate of viewing limits
yTop	Maximum y-coordinate.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetWindowViewportXform

```
#define INCL_GRE_XFORMS

BOOL GreGetWindowViewportXform (hdc, paXformData, pInstance, lFunction)
```

This function queries the current window or viewport transform matrix. On completion, paXformData is an array of two-dimensional values defining the current window or viewport transform matrix.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
paXformData	PXFORM	Pointer to return data in which the array of 6 matrix elements (M11, M12, M21, M22, M41, and M42) are to be stored
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetWindowViewportXform

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreIntersectClipRectangle

```
#define INCL_GRE_CLIP

LONG GreIntersectClipRectangle (hdc, prclRect, pInstance, lFunction)
```

This function sets the new clipping region to the intersection of the current clip region and the specified rectangle. When no clip region exists, GreIntersectClipRectangle must create one. The application must then free the handle when it subsequently selects another clip region. The return value of this function is the complexity of resultant DC region, which is defined as the intersection of all clipping.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclRect	PRECTL	Pointer to rectangle in world or device coordinates. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreIntersectClipRectangle.

prclRect RECTL structure, defined in world or device coordinates:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

All the boundaries of the rectangle are inclusive (part of the interior) and are not clipped. If COM_TRANSFORM is not set, the function expects the rectangle to be in device coordinates.

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the DC region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT Rectangular region
RGN_COMPLEX Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreModifyPath

```
#define INCL_GRE_PATHS

BOOL GreModifyPath (hdc, idPath, cmdMode, pInstance, lFunction)
```

This function modifies a path. When the transform is singular, GreModifyPath is invalid and causes an error to be logged.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
idPath	LONG	Path ID.
cmdMode	LONG	Modify mode. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreModifyPath.

cmdMode The only valid flag is:

MPATH_STROKE This replaces the original path (with a path that encloses the shape produced by stroking the original path) by using the current geometric wide line attribute. Any open figures within the path are not closed.

The envelope includes the effects of *line joins* and *line ends* according to the current values of these attributes. For example, where a line joins an arc, the common point is handled according to the line join attribute in the current line attribute bundle. (See "Line Attributes" on page 8-3.) When a figure is closed by using GreCloseFigure, the line join attribute in the current line bundle is applied to the start and end points. The envelope takes account of any crossings. For example, a stroked *X* does not have a hole in the middle if it is subsequently drawn in exclusive-OR mode.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE

PMERR_INV_COORDINATE
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_MODIFY_PATH_MODE
PMERR_INV_PATH_ID
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_TRANSFORM_TYPE
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When this function is performed on a path, the only subsequent operations allowed are GreFillPath (in winding mode) and GreSelectClipPath (in winding mode).

GreMultiplyXforms

```
#define INCL_GRE_XFORMS

BOOL GreMultiplyXforms (hdc, paXform, paNewXformData, lMode, pInstance, lFunction)
```

This function multiplies the transform matrix (defined by paNewXformData) by the corresponding matrix in paXform. The result is stored in paXform. When this function is used to make a series of matrix multiplications on the same matrix, some loss of accuracy can occur due to rounding because no higher precision can be retained across calls.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paXform	PXFORM	Pointer to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, and M42.
paNewXformData	PXFORM	Pointer to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, and M42.
lMode	LONG	Specifies how supplied array is used to set matrix. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreMultiplyXforms.

lMode Indicates how to use the array to set the matrix:

SX_UNITY	Set unity transform, ignore array values
SX_CAT_AFTER	Concatenate after
SX_CAT_BEFORE	Concatenate before
SX_OVERWRITE	Overwrite.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_MATRIX_ELEMENT
PMERR_INV_TRANSFORM_TYPE.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreOffsetClipRegion

```
#define INCL_GRE_CLIP

LONG GreOffsetClipRegion (hdc, pdpt, pInstance, lFunction)
```

This function moves the clipping region by the specified amount. The value returned is the complexity of the resultant DC region, that is, the intersection of all clipping (such as clip path, viewing limits, graphics field, clip region, and visible region).

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pdpt	PPOINTL	Pointer to offset by which clipping region is to be moved. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreOffsetClipRegion.

pdpt Offsets by which the clip region is to be moved in world coordinates:

dx X offset
dy Y offset.

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT Rectangular region
RGN_COMPLEX Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_OFFSET
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreOffsetRegion

```
#define INCL_GRE_REGIONS

BOOL GreOffsetRegion (hdc, hrgn, pdpt, pInstance, lFunction)
```

This function moves the given region by the specified amounts (unless the region is in use as a clipping region when an error is raised).

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgn	HRGN	Region handle.
pdpt	PPOINTL	Pointer to offset by which region is to be moved. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreOffsetRegion.

pdpt Offsets, in device coordinates:

dx X offset
dy Y offset.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_HRGN_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreOutlinePath

```
#define INCL_GRE_PATHS

LONG GreOutlinePath (hdc, idPath, flOptions, pInstance, lFunction)
```

This function draws the boundary of the path indicated by idPath. GreOutlinePath is also used to draw area boundaries.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
idPath	LONG	Path ID. Must be *1*.
flOptions	ULONG	All these flags are reserved.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreOutlinePath.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PATH_ID
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePaintRegion

```
#define INCL_GRE_REGIONS

LONG GrePaintRegion (hdc, hrgn, pInstance, lFunction)
```

This function paints the specified region by using the current area foreground and background colors. Mixing is controlled only by the area foreground mix. GrePaintRegion returns an error when the region is currently selected as a clip region.

If the preprocessor in the graphics engine receives a GrePaintRegion call for a null region, the preprocessor returns GPI_OK and does not forward the call through the dispatch table to the handling routine. Presentation drivers for devices that do not support BitBlt operations (for example, vector devices such as plotters) do not hook this function. Such drivers hook only the function to return an error code.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hrgn	HRGN	Region handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePaintRegion

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePartialArc

```
#define INCL_GRE_ARCS

LONG GrePartialArc (hdc, pptlCenter, fxMultiplier, fxStart, fxSweep, pInstance, lFunction)
```

This function draws a straight line from the current position to the starting point of a partial arc, and draws the specified partial arc. Upon completion, the current (x, y) position is the end of the partial arc. The dimensions of the full arc are defined as a multiplier, which is applied to the current arc parameters. The partial arc that is drawn is the section of the full arc that is enclosed by the specified start and sweep angles.

If GrePartialArc is used within a path or an area definition to continue a figure following a GreBox*xxx* or GreFullArc*xxx* function, the error PMERR_INV_NESTED_FIGURES is posted. This is because GreBox*xxx* and GreFullArc*xxx* generate a closed figure within an area or path definition.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pptlCenter	PPOINTL	Pointer to (x, y) coordinates for center of arc.
fxMultiplier	FIXED	Multiplier. See below.
fxStart	FIXED	Start angle that defines the starting point on the curve. See below.
fxSweep	FIXED	Sweep angle that defines the extent of the curve to be drawn. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePartialArc.

pptlCenter If COM_TRANSFORM is not set, the function expects the center of arc position to be in screen coordinates.

fxMultiplier This parameter defines the size of the full arc in relation to an arc drawn with the current arc parameters. The multiplier is a fixed-point value. The high-order WORD contains the integer portion; the low-order WORD contains the fractional portion. A value of 64KB gives a multiplier of *1*. There is an implementation limit of *255* for this value, which must not be negative.

fxStart/fxSweep Start and sweep angles are measured counterclockwise from the x-axis at the center of the arc before the arc parameters are applied. If the current arc parameters do not specify a circle, the angles are skewed to the same degree that the full arc is a skewed circle.

The angles are specified as doubleword values in fixed-point format. The high-order WORD contains the integer portion; the low-order WORD contains the fractional portion. A value of *6553* gives an angle of 1°. Both angles must be positive. Whether the arc is drawn in a clockwise or counterclockwise direction is determined by the arc parameters. An angle greater than 360° is also valid. In this case, after the initial line, a full arc is drawn followed by a partial arc of (lSweep MOD 360)°. See also GpiPartialArc in the *OS/2 2.0 Presentation Manager Programming Reference*.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK	Successful
GPI_HITS	Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_ANGLE_PARM
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MULTIPLIER
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePolyFillet

```
#define INCL_GRE_ARCS

LONG GrePolyFillet (hdc, paptlPoint, cPoints, pInstance, lFunction)
```

This function draws a fillet on a series of connected lines with the first line starting at the current (x, y) position. Upon completion, the current (x, y) position is set to the last point in the series.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paptlPoint	PPOINTL	Pointer to coordinates array. See below.
cPoints	LONG	Number of coordinate pairs. If this value is passed as *0*, the handling routine does nothing and returns Successful.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePolyFillet.

paptlPoint An array of cPoints (x, y) pairs, which contain the (x, y) coordinates of the end points for the lines. If COM_TRANSFORM is not set, the function expects the points to be in screen coordinates.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The shape of the fillet is controlled by a set of coordinates for a series of two or more connected lines. The fillet is tangential to the start point of the first line and to the end point of the last line. When more than two sets of coordinates are supplied, the fillet passes through the mid-points of the intermediate lines. An individual fillet always lies within the area bounded by the start, end, and control points. See GpiPolyFillet in the *OS/2 2.0 Presentation Manager Programming Reference* for more information.

GrePolyFilletSharp

```
#define INCL_GRE_ARCS

LONG GrePolyFilletSharp (hdc, paptlPoint, cPoints, pfxSharp, pInstance, lFunction)
```

This function draws a sequence of one or more sharp fillets starting at the current (x, y) position. As each fillet is drawn, the end point for the fillet becomes the start point for the next fillet. Upon completion, the current (x, y) position is the end point of the last fillet. Each fillet is controlled by a set of coordinates for two connected lines and by a sharpness value. The fillet is tangential to the start point of the first line and to the end point of the second line.

The sharpness value is determined from:

```
Sharpness = WD/DB
where:
       W is the mid-point of the line joining the end points
       B is the control point above the top of the fillet
       D is the point where the fillet intersects the line WC.
```

See GpiPolyFillet in the *OS/2 2.0 Presentation Manager Programming Reference* for more detail.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paptlPoint	PPOINTL	Pointer to Coordinates array. See below.
cPoints	LONG	Number of coordinate pairs.
pfxSharp	PFIXED	Pointer to Sharpness array. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePolyFilletSharp.

paptlPoint An array of cPoints (x, y) pairs. These are groups of four elements where each group contains two pairs of (x, y) coordinates for the control and end points of the fillets. If COM_TRANSFORM is not set, the function expects the points to be in screen coordinates.

cPoints When this is passed as *0*, the function does nothing and returns Successful.

pfxSharp An array of sharpness values with one element for each fillet. *Sharpness* is defined as a fixed-point value. The high-order WORD contains the integer portion; the low-order WORD contains the fractional portion. A value of 64KB gives a sharpness of *1*.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful

GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)

GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePolySpline

```
#define INCL_GRE_ARCS

LONG GrePolySpline (hdc, paptlPoint, cPoints, pInstance, lFunction)
```

This function draws a sequence of one or more Bezier splines starting at the current (x, y) position. As each spline is drawn, the specified end point for the spline becomes the start point for the next spline. Upon completion, the current (x, y) position is the end point of the last spline.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paptlPoint	PPOINTL	Pointer to coordinates array. See below.
cPoints	LONG	Number of coordinate pairs in the array.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePolySpline.

paptlPoint An array of (3 × cPoints)(x, y) values in groups of six elements where each group contains three pairs of (x, y) coordinates for the control and end points for the splines. If COM_TRANSFORM is not set, the function expects the points to be in screen coordinates.

cPoints When this is passed as *0*, the function does nothing and returns Successful.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH

PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The shape of each spline is controlled by a set of coordinates for three connected lines. The spline starts at the current position and ends at the end point of the third line. The end points of the first and second lines are used as control points. An individual spline always lies within the area bounded by the start, end, and control points. See GpiPolySpline in the *OS/2 2.0 Presentation Manager Programming Reference* for more information.

GrePolygonSet

```
#define INCL_GRE_LINE

LONG GrePolygonSet (hdc, flModel, flOptions, paPolygon, cPolygons, pInstance, lFunction)
```

This function draws a set of closed polygons. The polygons are filled using the current AREABUNDLE structure values. For the first polygon, the current position is the first point. For all subsequent polygons, all points that define the polygon are given explicitly. The polygons are automatically closed, if necessary, by drawing a line from the last vertex to the first. Notice that polygons can overlap, if needed.

Note: GrePolygonSet is not valid when the COM_AREA or COM_PATH flag is set.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
flModel	LONG	Model flags. See below.
flOptions	ULONG	Option flags. See below.
paPolygon	PPOLYGON	Pointer to an array of POLYGON structures. See below.
cPolygons	LONG	Number of POLYGON structures. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD=flags; Low-order WORD = NGrePolygonSet.

flModel These flags determine how polygons are drawn:

POLYGON_INCL Default setting. Fill is inclusive of bottom right.
POLYGON_EXCL Fill is exclusive of bottom right. Aids migration from other graphics models.

flOptions These flags determine how the polygons are filled:

POLYGON_ALTERNATE Fill is performed by using the alternate mode.
POLYGON_WINDING Fill is performed by using the winding mode.
POLYGON_BOUNDARY Draw boundary lines.
POLYGON_NOBOUNDARY Do not draw boundary lines.

paPolygon Pointer to an array of POLYGON structures:

ulPoints Number of points in this polygon
aPointl Array of points defining polygon

If COM_TRANSFORM is not set, the function expects the points to be in screen coordinates.

cPolygons When this is passed as *0*, the handling routine takes no action except to return Successful.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePtInRegion

```
#define INCL_GRE_REGIONS

LONG GrePtInRegion (hdc, hrgn, pptlPoint, pInstance, lFunction)
```

This function checks whether a point lies within a region. GrePtInRegion raises an error when hrgn is the handle of the currently selected clip region. The point is always expected in device coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hrgn	HRGN	Region handle
pptlPoint	PPOINTL	Pointer to (x, y) point specified in device coordinates
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePtInRegion

Return Codes: This function returns an integer (lInside) indicating whether the point is inside the region:

RGN_ERROR Error
PRGN_INSIDE In region
PRGN_OUTSIDE Not in region.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePtVisible

```
#define INCL_GRE_CLIP

LONG GrePtVisible (hdc, pptlPoint, pInstance, lFunction)
```

This function checks whether a point is visible within the DC region of the specified device context. The *DC region* is defined as the intersection of the application clipping and the window clipping.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pptlPoint	PPOINTL	Pointer to (x, y) point in world or screen coordinates
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePtVisible

pptlPoint If COM_TRANSFORM is not set, the function expects the point to be in screen coordinates.

Return Codes: This function returns an integer (lVisible) indicating the visibility of the point:

PVIS_ERROR Error
PVIS_INVISIBLE Not visible
PVIS_VISIBLE Visible.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryClipRegion

```
#define INCL_GRE_CLIP

HRGN GreQueryClipRegion (hdc, pInstance, lFunction)
```

This function returns the handle of the currently selected clip region, or NULL (if none exists).

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryClipRegion

Return Codes: This function returns the handle of the clip region:

hrgn Region handle
NULL Null handle (no region selected)
HRGN_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryHWPaletteInfo

```
#define INCL_GRE_PALETTE

DDIENTRY GreQueryHWPaletteInfo (hdc, ulStart, cclr, pclr, pInstance, lFunction)
```

This function fills a buffer with all the information for the hardware palette. If cclr=*0*, only the size required for the buffer is returned. The information returned is the same as that which is used to create a palette. This allows a caller to use the returned information directly to create the same palette in another context by using the first four DWORDs directly and a pointer to the color entries.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
ulStart	ULONG	Starting index, first palette entry to query
cclr	ULONG	Count of palette entries
pclr	PULONG	Pointer to table of RGB2 palette entries
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD=flags; low-order WORD=NGreQueryHWPaletteInfo

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_PALETTE.

Remarks: The existing color table GreQuery*xxx* APIs are still supported.

GreQueryPaletteRealization

```
#define INCL_GRE_PALETTE

DDIENTRY GreQueryPaletteRealization (hdc, ulStart, cclr, pclr, pInstance, lFunction)
```

This function returns the mapping from the logical palette to the HW palette as an array of ULONGs. GreQueryPaletteRealization gives applications the ability to predict the outcome of color mixing operations.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
ulStart	ULONG	Starting index, first palette entry to query
cclr	ULONG	Count of palette entries
pclr	PULONG	Pointer to table of RGB2 palette entries
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryPaletteRealization

Return Codes: This function returns the count of the palette entries.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_DC.

GreRealizePalette

```
#define INCL_GRE_PALETTE

DDIENTRY GreRealizePalette (hdc, pflType, pcSlotsChanged, pInstance, lFunction)
```

This function is called by the Window Manager during processing of a call to WinRealizePalette. GreRealizePalette takes a logical palette and assigns it slots in the hardware palette.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pflType	PULONG	Specifies whether the DC is a foreground or background window. Used to indicate if the default colors have changed. See below.
pcSlotsChanged	PULONG	Pointer to the returned count of hardware slots changed.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRealizePalette.

pflType Specifies whether the DC represents a foreground or background window. The possible values are:

RP_BACKGROUND = 0 When the presentation driver receives this call for a background DC, it constructs the logical-to-physical Index Translate table for the DC's palette (by using available slots and the nearest mapping to used, non-animating slots), and marks the translate table as *clean*.

RP_FOREGROUND = 1 When the presentation driver receives this call for a foreground DC, it sets the DC's selected palette into the physical palette. It also marks all the translate tables for the other logical palettes as *dirty*.

RP_DEFAULTSCHANGED = 2 Returned by the presentation driver to indicate that all direct DCs need repainting.

Return Codes: On completion, this function returns the following value:

cclr Number of mappings that changed.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_INV_DC
PMERR_NO_PALETTE
PMERR_PAL_ERROR.

Remarks: If a palette has not been explicitly selected into the DC before a call to GreRealizePalette is made, the default palette is implicitly in effect and is realized. This function requires no action for memory DCs. A call to GreRealizePalette is a real operation only when an actual device with a palette is referred to. It does not return an error for memory DCs, therefore, an application need not be concerned with the DC type if it has a number of DCs with palettes.

When GreBitblt is performed between a memory DC and a device DC, the presentation driver has to do the appropriate translation between the memory bit map's color table or logical palette and the physical palette.

GreRectInRegion

```
#define INCL_GRE_REGIONS

LONG GreRectInRegion (hdc, hrgn, prclRect, pInstance, lFunction)
```

This function checks whether any part of a given rectangle lies within the specified region. GreRectInRegion raises an error if hrgn is the handle of the currently selected clip region.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgn	HRGN	Region handle.
prclRect	PRECTL	Pointer to rectangle in device coordinates. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRectInRegion.

prclRect RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

The rectangle passed is considered *exclusive* on the right and top sides. If the rectangle represents the bounding box of a graphic object to be drawn, the drawing might include the right and top sides. In this case, the caller should *inflate* the top and right sides by one unit in device-coordinate space.

Return Codes: This function returns an integer (lInside) indicating whether the rectangle is inside the region:

RRGN_ERROR Error
RRGN_INSIDE All in region
RRGN_PARTIAL Partially in region
RRGN_OUTSIDE Not in region.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreRectVisible

```
#define INCL_GRE_CLIP

LONG GreRectVisible (hdc, prclRect, pInstance, lFunction)
```

This function checks whether any part of the given rectangle is visible within the DC region. The DC region is the intersection of the application clipping and the window clipping.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclRect	PRECTL	Pointer to rectangle in world or screen coordinates. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRectVisible.

prclRect RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

The rectangle passed is considered *exclusive* on the right and top sides. If the rectangle represents the bounding box of a graphic object to be drawn, the drawing might include the right and top sides. In this case, the caller should *inflate* the top and right sides by one unit in screen-coordinate space. This factor can be difficult to control in world coordinates. If precision is required, use screen coordinates.

Return Codes: This function returns an integer (lVisible) indicating the visibility of the rectangle.

RVIS_ERROR Error
RVIS_INVISIBLE Not visible
RVIS_PARTIAL Partially visible
RVIS_VISIBLE All visible.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreRegionSelectBitmap

```
#define INCL_GRE_CLIP

BOOL GreRegionSelectBitmap (hdc, hbm, pInstance, lFunction)
```

This function is called when a new bit-map handle is selected into a memory DC. It removes the old visible region and informs the presentation driver that the DC region must be recalculated.

Note: In a nondisplay DC, the visible region represents the device boundary.

Support: This function is supported by the engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hbm	HBITMAP	Handle of bit map to be selected
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRegionSelectBitmap

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreRestorePath

```
#define INCL_GRE_PATHS

BOOL GreRestorePath (hdc, cSave, pInstance, lFunction)
```

This function is called during RestoreDC and CloseDC to allow the path handling routines to restore their local data structures. When a DC is closed, GreRestorePath is called with a count of *0* to free its local data.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cSave	LONG	DC save level. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRestorePath.

cSave Indicates the saved DC state, which the handling routine uses to restore the path. When cSave is passed as *–1*, the handling routine resets the path to its initial state (*–1* is passed to this routine from GreResetDC and is the only valid negative value). A value of *0* indicates that the path is restored to its initial state. Other positive values identify which saved level is restored. See "Enable Subfunction 08H – RestoreDCState" on page 7-18.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function is required to ensure that memory is freed.

GreRestoreRegion

```
#define INCL_GRE_CLIP

BOOL GreRestoreRegion (hdc, cSave, pInstance, lFunction)
```

This function is called during RestoreDC and CloseDC to allow the region handling routines to restore their local data structures. When a DC is closed, GreRestoreRegion is called with a count of *0* to free its local data. The clip region currently selected into the DC is deleted by this function.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cSave	LONG	DC save level. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRestoreRegion.

cSave Indicates the saved DC state, which the handling routine uses to restore the region. When this is passed as *–1*, the handling routine resets the region to its initial state (*–1* is passed to this routine from GreResetDC and is the only valid negative value). A value of *0* indicates that the region is restored to its initial state. Other positive values identify which saved level is restored. See "Enable Subfunction 08H – RestoreDCState" on page 7-18.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function is required to ensure that memory is freed.

GreRestoreXform

```
#define INCL_GRE_XFORMS

BOOL GreRestoreXform (hdc, cSave, pInstance, lFunction)
```

This function is called during RestoreDC and CloseDC to allow simulations to restore their local data structures. When a DC is closed, GreRestoreXform is called with a count of *0* to free its local data.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cSave	LONG	DC save level. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRestoreXform.

cSave Indicates the saved DC state, which the handling routine uses to restore the transform. When cSave is passed as *–1*, the handling routine resets the transform to its initial state (*–1* is passed to this routine from GreResetDC and is the only valid negative value). A value of *0* indicates that the transform is reset to its initial state. Other positive values identify which saved level is restored. See "Enable Subfunction 08H – RestoreDCState" on page 7-18.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreRestoreXformData

```
#define INCL_GRE_XFORMS

BOOL GreRestoreXformData (hdc, ulSize, pBuffer, pInstance, lFunction)
```

This function restores a previously saved transform state. The current transform state is overwritten.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
ulSize	ULONG	Size in bytes of pBuffer
pBuffer	PBYTE	Pointer to stored transform data
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRestoreXformData

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PICK_APERTURE_POSN
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSavePath

```
#define INCL_GRE_PATHS

BOOL GreSavePath (hdc, cSave, pInstance, lFunction)
```

This function is called during SaveDC and OpenDC to allow the path handling routines to save their local data structures. When a new DC is created, GreSavePath is called with a count of *1* to initialize its local data. A Save with a count of one more than the current save level generates intervening levels.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cSave	LONG	DC save level. This must not be a negative value.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSavePath.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function is required to ensure that the path data is saved.

GreSaveRegion

```
#define INCL_GRE_CLIP

BOOL GreSaveRegion (hdc, cSave, pInstance, lFunction)
```

This function is called during SaveDC and OpenDC to allow the region handling routines to save their local data structures. When a new DC is created, GreSaveRegion is called with a save level of *1* to initialize its local data. A Save with a count of one more than the current save level generates intervening levels. A negative save value is invalid. When initializing a new DC, a valid visible region handle must be created.

Support: This function is supported by the graphics engine, it can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cSave	LONG	DC save level. This must not be a negative value.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSaveRegion.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_RECT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function is required to ensure that the region data is saved.

GreSaveXform

```
#define INCL_GRE_XFORMS

BOOL GreSaveXform (hdc, cSave, pInstance, lFunction)
```

This function is called during SaveDC and OpenDC to allow simulations to save their local data structures. A Save with a count of one more than the current save level generates intervening levels. When a new DC is created, GreSaveXform is called with a count of *1* to initialize its local data.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
cSave	LONG	DC save level. This must not be a negative value.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSaveXform.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When initializing a new DC, the default transform must be set.

GreSaveXformData

```
#define INCL_GRE_XFORMS

ULONG GreSaveXformData (hdc, ulSize, pBuffer, pInstance, lFunction)
```

This function stores the current transform state in pBuffer.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
ulSize	ULONG	Size in bytes required for pBuffer. See below.
pBuffer	PBYTE	Pointer to an area where the data is to be stored.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSaveXformData.

ulSize Can be specified as *0*. In this case, the function returns the size of the transform data:

```
ulSize=GreSaveXformData (hdc, 0, 0);        /* Find out how large the buffer must be, */
GreSaveXformData (hdc, ulSize, pBuffer);    /* then save the transform state.         */
```

Return Codes: When called with ulSize specified as *0*, this function returns the size of the buffer required to save the transform state. Otherwise, it returns GPI_OK to indicate a successful completion. In either case, GPI_ERROR is returned to indicate failure.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSelectClipPath

```
#define INCL_GRE_PATHS

BOOL GreSelectClipPath (hdc, idPath, flOptions, pInstance, lFunction)
```

This function resets or modifies the currently selected clip path. Before modifying the clip path, the handling routine must close all open figures in the path. The modified clip path consists of the areas common to both the old clip path and the figures in the specified path. When the constituent paths are defined, the modified clip path is bound in device coordinates and is used for all subsequent drawing, including filling. All of the boundaries of the area are considered to be part of the interior and are not clipped. Upon completion, the handling routine deletes the old clip path indicated by idPath, allowing a new path definition, which can co-exist with the new clip path, to begin as required. For more information, see GpiSetClipPath in the *OS/2 2.0 Presentation Manager Programming Reference*.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
idPath	LONG	Path ID. See below.
flOptions	ULONG	Option flags. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSelectClipPath.

idPath When this is passed as *0*, the path is reset to *no clipping*. Otherwise, the new clip path is intersected with the existing clip path.

flOptions The following flags determine the path mode:

SCP_ALTERNATE (Default.) Use alternate mode for intersection.
SCP_WINDING Use winding mode for intersection.

The following flags determine whether the new path replaces or intersects the initial path:

SCP_AND Intersect the two clip paths. If idPath is not *1*, this flag is invalid and the handling routine must log an error.

SCP_RESET (Default). Replace the initial clip path. If idPath is not *0*, this flag is invalid and the handling routine must log an error.

Other flags are reserved and must be *0*.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CLIP_PATH_OPTIONS
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PATH_ID
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSelectClipRegion

```
#define INCL_GRE_CLIP

LONG GreSelectClipRegion (hdc, hrgn, phrgnOld, pInstance, lFunction)
```

This function specifies the region to be used for clipping. A region can only be selected by one DC at a time. Once a region is selected, operations that reference its handle are invalid. The coordinates of the region are in device coordinates within the DC. Clipping is inclusive at the left and bottom boundaries, exclusive at the right and top boundaries. Functions that modify the clipping region also modify the region when its handle is returned by a subsequent GreSelectClipRegion.

GreSelectClipRegion passes the handle of the previously selected clip region back to the calling routine at the location addressed by phrgnOld. If this handle is not NULL, the calling routine is responsible for deleting the previous clip region. If the default clip region is in use when GreSelectClipRegion is called, a null region handle is returned.

Note: The clip region should be deleted whether it was created explicitly, or in a call to GreIntersectClipRectangle or GreExcludeClipRectangle.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgn	HRGN	Region handle. See below.
phrgnOld	PHRGN	Pointer to old region handle.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSelectClipRegion.

hrgn When NULL, the clip region is set to *no clipping* (the initial state).
phrgnOld Previously selected region. If this is a NULL handle, there was no clip region.

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the DC region:

RGN_ERROR Error
RGN_NULL Null region
RGN_RECT Rectangular region
RGN_COMPLEX Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSelectPathRegion

```
#define INCL_GRE_CLIP

BOOL GreSelectPathRegion  (hdc, hrgn, pInstance, lFunction)
```

This function merges a region representing a path into the DC region. A *path region* behaves exactly like any other region. It has no special properties relating to paths.

Support: This function is supported by the graphics engine.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hrgn	HRGN	Region handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSelectPathRegion

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetArcParameters

```
#define INCL_GRE_ARCS

BOOL GreSetArcParameters (hdc, pArcParms, pInstance, lFunction)
```

This function sets the arc parameters.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
pArcParms	PARCPARAMS	Pointer to an array containing the arc parameters. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetArcParameters.

pArcParms ARCPARAMS structure:

lP P coefficient
lQ Q coefficient
lR R coefficient
lS S coefficient.

See the *OS/2 2.0 Presentation Manager Programming Reference* for a full description of the arc parameters structure.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_HDC.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetGlobalViewingXform

```
#define INCL_GRE_XFORMS

BOOL GreSetGlobalViewingXform (hdc, paXformData, flOptions, pInstance, lFunction)
```

This function sets the global viewing transform matrix elements.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paXformData	PXFORM	Pointer to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, and M42.
flOptions	LONG	Specifies how the supplied array is used to set the matrix. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetGlobalViewingXform.

flOptions Valid values are:

SX_UNITY	Set unity transform. Ignore array values.
SX_CAT_AFTER	Concatenate after.
SX_CAT_BEFORE	Concatenate before.
SX_OVERWRITE	Overwrite.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT

PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_TRANSFORM_TYPE
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetGraphicsField

```
#define INCL_GRE_XFORMS

BOOL GreSetGraphicsField (hdc, prclGraphicsField, pInstance, lFunction)
```

This function sets the boundaries of the graphics field (clip) limits in page-coordinate space. The boundaries are inclusive so that points on them are not clipped. By default, no clipping is performed.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclGraphicsField	PRECTL	Pointer to graphics field. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetGraphicsField.

prclGraphicsField RECTL structure:

xLeft Minimum x-coordinate of graphics field
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of graphics field
yTop Maximum y-coordinate.

An error is raised when xLeft is greater than xRight, or yBottom is greater than yTop.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_COORDINATE
PMERR_INV_GRAPHICS_FIELD
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetModelXform

```
#define INCL_GRE_XFORMS

BOOL GreSetModelXform (hdc, paXformData, flOptions, pInstance, lFunction)
```

This function sets the model transform matrix elements as specified.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paXformData	PXFORM	Pointer to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, and M42.
flOptions	LONG	Specifies how the supplied array is used to set the matrix. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetModelXform.

flOptions Valid values are:

SX_UNITY	Set unity transform. Ignore array values.
SX_CAT_AFTER	Concatenate after.
SX_CAT_BEFORE	Concatenate before.
SX_OVERWRITE	Overwrite.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT

PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_TRANSFORM_TYPE
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetPageUnits

```
#define INCL_GRE_XFORMS

BOOL GreSetPageUnits (hdc, ulUnits, lWidth, lHeight, pInstance, lFunction)
```

This function sets the page units controlling the device transform.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
ulUnits	ULONG	Page units. See below.
lWidth	LONG	Page width (w). See below.
lHeight	LONG	Page height (h). See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetPageUnits.

ulUnits Page units, as:

PU_ARBITRARY Isotropic, arbitrary units defined by lHeight and lWidth. The page viewport is constructed to give equal x and y spacing on the physical device with at least one dimension of the page completely filling the corresponding default device dimension (that is, maximized window size and paper size). The origin is at the bottom left.

PU_PELS Pel coordinates with the origin at the bottom left.

PU_LOMETRIC Low resolution metric. These are units of 0.1mm with the origin at the bottom left.

PU_HIMETRIC High resolution metric. These are units of 0.01mm with the origin at the bottom left.

PU_LOENGLISH Units of 0.01 inch with the origin at the bottom left.

PU_HIENGLISH Units of 0.001 inch with the origin at the bottom left.

PU_TWIPS Twentieths of an imperial point. These are units of 1/1440 inch with the origin at the bottom left.

Other bits are reserved, and must be preserved and returned by GetPageUnits.

lWidth For PU_ARBITRARY. When either lWidth or lHeight is passed as *0*, that dimension is set to produce equal x and y spacing on the physical device with both dimensions completely filling the default device dimensions. When both are passed as *0*, then they are set to produce equal x and y spacing on the physical device.

lWidth and lHeight completely fill the default device dimensions so that one is equal to the corresponding default device dimension, and the other is equal to, or greater than, its corresponding default device dimension. These are measured in pels. For other units, a value of *0* for *w* or *h* causes the page to be set to the corresponding default dimension (that is, maximized window size and paper size) in page units or pels for isotropic units.

lHeight See lWidth.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_RECT
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: This function causes the Window/Viewport Transform, Page Viewport, and Device Transform matrixes to be updated as shown below. For PU_LOMETRIC, PU_HIMETRIC, PU_TWIPS, PU_LOENGLISH, PU_HIENGLISH, and PU_PELS:

Window/Viewport Transform	Unity
Page Viewport	(0,0) (sx*lWidth−1, sy*lHeight−1)
Device Transform	As defined by the mapping from Page Window to Page Viewport

Where *sx* is the horizontal scaling required by page units for the device (or *1* for PU_PELS), and *sy* is the vertical scaling required by page units for the device (or *1* for PU_PELS).

For PU_ARBITRARY:

Window/Viewport Transform	Unity
Page Viewport	(0,0) (X2,Y2)
Device Transform	As defined by the mapping from Page Window to Page Viewport

Where *Dh* is the default device (maximized window) height in pels, *Dw* is the default device (maximized window) width in pels, and *Par* is the pel (width/height) aspect ratio. X2 and Y2 are integers, determined as follows:

```
When (lWidth/lHeight) > Par*(Dw/Dh):
      X2 = Dw-1
      Y2 = Par*Dw * (lWidth/lHeight)-1

When (lWidth/lHeight) < Par*(Dw/Dh):
      X2 = (1/Par) * Dh * (lWidth/lHeight)-1
      Y2 = Dh-1

Otherwise, when (lWidth/lHeight) = Par*(Dw/Dh):
      X2 = Dw-1
      Y2 = Dh-1
```

GreSetPageViewport

```
#define INCL_GRE_XFORMS

BOOL GreSetPageViewport (hdc, prclViewport, flOptions, pInstance, lFunction)
```

This function sets the page viewport in device coordinates so that the graphics engine can update the device transform by using the page window and page viewport coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclViewport	PRECTL	Pointer to page viewport boundaries in device coordinates. See below.
flOptions	ULONG	Reserved parameter. The only valid value is *0*.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetPageViewport.

prclViewport RECTL structure:

xLeft Minimum x-coordinate of viewport
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of viewport
yTop Maximum y-coordinate.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT

PMERR_INV_PAGE_VIEWPORT
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SET_VIEWPORT_OPTION
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetRectRegion

```
#define INCL_GRE_REGIONS

BOOL GreSetRectRegion (hdc, hrgn, parcRgn, cRect, pInstance, lFunction)
```

This function sets the specified region to the region definition given by an array of rectangles unless the region is in use as a clipping region (in which case, an error is raised). When no rectangles are specified (that is, cRect is *0*), GreSetRectRegion produces an empty region.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
hrgn	HRGN	Region handle
parcRgn	PRECTL	Pointer to region definition. See below.
cRect	LONG	Count of rectangles in the region definition.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetRectRegion.

parcRgn The region is defined as an array of rectangles. Each rectangle in the array is defined by a RECTL structure:

xLeft Minimum x-coordinate of rectangle
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of rectangle
yTop Maximum y-coordinate.

The region is defined by the OR of all the rectangles. For each rectangle, xRight must be equal to, or greater than, xLeft. yTop must be equal to, or greater than, yBottom. The bottom and left boundaries of each rectangle are part of the interior of the region, the top and right boundaries are not. If COM_TRANSFORM is not set, the function expects rectangles to be in device coordinates.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HRGN
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetViewingLimits

```
#define INCL_GRE_XFORMS

BOOL GreSetViewingLimits (hdc, prclViewingLimits, pInstance, lFunction)
```

This function sets the boundaries of the viewing (clip) limits (in model space) to the specified values. The boundaries are inclusive and are not clipped. The viewing-limits coordinates are transformed to make a clipping rectangle in page-coordinate or device-coordinate space. Any rotation or shear of this rectangle is ignored. When the left boundary is greater than the right, or the bottom boundary is greater than the top, a NULL rectangle is defined and all points are clipped.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
prclViewingLimits	PRECTL	Pointer to limits of viewing area. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetViewingLimits.

prclViewingLimits RECTL structure:

xLeft	Minimum x-coordinate of viewing limits
yBottom	Minimum y-coordinate
xRight	Maximum x-coordinate of viewing limits
yTop	Maximum y-coordinate.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_HDC_BUSY
PMERR_INV_COORDINATE
PMERR_INV_GRAPHICS_FIELD
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetWindowViewportXform

```
#define INCL_GRE_XFORMS

BOOL GreSetWindowViewportXform (hdc, paXformData, flOptions, pInstance, lFunction)
```

This function sets the window or viewport transform matrix elements as specified.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
paXformData	PXFORM	Pointer to an array of 6 matrix elements for two-dimensional formation. These are M11, M12, M21, M22, M41, and M42.
flOptions	LONG	Specifies how the supplied array should be used to set the matrix. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetWindowViewportXform.

flOptions Valid values are:

SX_UNITY	Set unity transform. Ignore array values.
SX_CAT_AFTER	Concatenate after.
SX_CAT_BEFORE	Concatenate before.
SX_OVERWRITE	Overwrite.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT

PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_TRANSFORM_TYPE
PMERR_PATH_LIMIT_EXCEEDED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetXformRect

```
#define INCL_GRE_CLIP

LONG GreSetXformRect (hdc, prclRect, pInstance, lFunction)
```

This function intersects a rectangle in device coordinates with the DC region. The rectangle is inclusive at the bottom and left boundaries, exclusive at the top and right boundaries.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
prclRect	PRECTL	Pointer to a rectangle in device coordinates
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetXformRect

prclRect RECTL structure:

- **xLeft** Minimum x-coordinate of rectangle
- **yBottom** Minimum y-coordinate
- **xRight** Maximum x-coordinate of rectangle
- **yTop** Maximum y-coordinate.

Return Codes: This function returns an integer (lComplexity) indicating the complexity of the DC region:

- **RGN_ERROR** Error
- **RGN_NULL** Null region
- **RGN_RECT** Rectangular region
- **RGN_COMPLEX** Complex region (more than 1 rectangle).

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetupDC

```
#define INCL_GRE_CLIP

BOOL GreSetupDC (hdc, hrgnVis, xOrg, yOrg, prclBounds, flOptions, pInstance, lFunction)
```

This function initializes the device context to the region determined by flOptions.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
hrgnVis	HRGN	Visible region handle
xOrg	LONG	X-coordinate of DC origin, specified in screen coordinates
yOrg	LONG	Y-coordinate of DC origin, specified in screen coordinates
prclBounds	PRECTL	Bounding rectangle in device coordinates
flOptions	ULONG	Option flags. See below.
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetupDC

flOptions These flags determine the region returned in prclBounds:

SETUPDC_VISRGN	Replace the contents of the visible region of hdc with the contents of hrgnVis.
SETUPDC_ORIGIN	Set the DC origin to (xOrg, yOrg).
SETUPDC_ACCUMBOUNDSON	Turn on bounds accumulation. This only affects the COM_ALT_BOUND flag. If COM_ALT_BOUND (see 1-6) is not set, the bounds rectangle is reset to an empty rectangle. If COM_ALT_BOUND is already set, the bounds rectangle is not changed.
SETUPDC_ACCUMBOUNDSOFF	Turn off bounds accumulation.
SETUPDC_REALCLIP	Recalculate the true device clipping region. This bit is normally set, but can be *0* when immediate recalculation is not required.
SETUPDC_SETOWNER	When this bit is set, the DC must belong to the current process.
SETUPDC_CLEANDC	When this bit is set, the simulation marks the visible regions as valid and calls GreNotifyClipChange in the presentation driver.

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreStrokePath

```
#define INCL_GRE_PATHS

LONG GreStrokePath (hdc, idPath, flOptions, pInstance, lFunction)
```

This function converts a path to the envelope of a wide line by using the current geometric line attribute (see "Line Attributes" on page 8-3). The converted path is filled by using winding mode and the current area attributes (see "Area (Pattern) Attributes" on page 8-5), and then drawn. When the path has been drawn, it is deleted. Notice that GreStrokePath is equivalent to GreModifyPath followed by GreFillPath. It is provided to allow the presentation driver to optimize its storage.

Note: GreModifyPath and GreStrokePath are the only functions that can construct geometric wide lines.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle.
idPath	LONG	Path ID. The only valid value is *1*.
flOptions	ULONG	Reserved. Must be *0*.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreStrokePath.

Return Codes: This function returns an integer (cHits) indicating, where appropriate, whether correlation hits were detected:

GPI_OK Successful
GPI_HITS Successful with correlate hit (returned by display drivers when the correlate flag is *on*, and a hit is detected)
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_INDEX
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_FILL_PATH_OPTIONS
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH

PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_MODIFY_PATH_MODE
PMERR_INV_PATH_ID
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PICK_APERTURE_POSN
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_TRANSFORM_TYPE
PMERR_PATH_LIMIT_EXCEEDED
PMERR_PATH_UNKNOWN
PMERR_REGION_IS_CLIP_REGION.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreUpdateColors

```
#define INCL_GRE_PALETTE

DDIENTRY GreUpdateColors (hdc, pInstance, lFunction)
```

This function causes a pel-by-pel remapping of the DCs visible area on the screen, and is typically called when an application receives notification from the Window Manager that the physical palette has changed. This indicates that the window colors might have changed to incorrect colors if the window is in the background. By calling GreUpdateColors, the pels are changed to the color in the current physical palette that comes closest to the desired color.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameter	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreUpdateColors

Return Codes: On completion, this function returns the following values:

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the handling routine must call WinSetErrorInfo to post the condition. Error codes for conditions that the handling routine is expected to check include:

PMERR_ERROR_ZERO
PMERR_INV_DC
PMERR_NO_PALETTE.

Chapter 11. Graphics Engine Internal Functions

The handling routines for these functions are in the graphics engine. Because they are not called through the dispatch table, they cannot be hooked by the presentation driver. Descriptions of these internal functions are provided. The functions are grouped according to the conditional include sections of the header file:

- Device context functions (INCL_GRE_DCS)
- Device support functions (INCL_GRE_DEVSUPPORT)
- Font functions (INCL_GRE_FONTS)
- Journal functions (INCL_GRE_JOURNALING)
- LCID functions (INCL_GRE_LCID)
- Set ID functions (INCL_GRE_SETID).

Each description shows what the function does, the parameters passed to the engine and the values that the function returns.

Font Functions

When Presentation Manager is initialized, the engine is responsible for establishing the default image and outline fonts, and the default pattern and marker sets. The presentation driver can supply device fonts as default fonts. If this is done, the driver can request the graphics engine to manage the device fonts by setting the CAPS_FONT_OUTLINE_MANAGE or CAPS_FONT_IMAGE_MANAGE flag in the array of device capabilities (see "GreQueryDeviceCaps" on page 8-111). When these flags are set, the engine performs any mapping and transforms that might be necessary. To use a default font belonging to the presentation driver, the font handle is passed to the engine in response to GreQueryDevResource (see page 8-113).

Font Matching: When requested by an application, the mapping of a font to a loaded font that is physically available is done by the graphics engine. A *physical font* can be one that is built into a particular device or a public font loaded from a FON file when Presentation Manager is initialized. The engine matches fonts by checking the values for szFacename and lMatch (in the font metrics structures) against those passed to it in GreCreateLogicalFont. If no match is found, the engine proceeds as if the match number was passed with a value of *0*. When a value of *0* is passed for either szFaceName or lMatch, the engine ignores that parameter and considers it to match anything. If no face name is specified, the engine supplies a default font from the presentation driver (see "GreQueryDevResource" on page 8-113), or if none is available, maps the requested font to one of its own default fonts.

When a face name is specified and the match number is *0*, the engine checks to see if an image font was requested and, if true, then searches for a font of the specified dimensions. If this search fails, the engine then attempts to match the font to an outline font with the face name and selection flags requested. When the match number is *0* and the required font is an outline font, the engine searches for an outline font. Should all these searches fail, a default font is used. See "GreRealizeFont" on page 8-130.

Journaling Functions (Hardcopy Drivers Only)

Journal functions allow presentation drivers to optimize the processing of data for raster devices such as dot-matrix or laser printers, which print bands of data. The size of a band is determined by the type of device and the amount of memory the driver can use to build its bit map. As an example, a color laser printer might need the full 24 bits per pel. In this case, several bands may be needed to make a page. A simple dot matrix printer that uses 1 bit per pel could treat the whole page as a single band. Refer to "Banding" for a description of how a presentation driver would use journaling functions to perform the technique of banding.

The hardcopy driver creates a journal file when the DC is enabled and starts recording in response to GreEscape (DEVESC_STARTDOC). The graphics engine stores the Gre*xxx* functions in the journal file as they are passed to the DC until told to stop recording by the hardcopy driver.

If the presentation driver passes the bit flag JNL_DRAW_OPTIMIZATION when it calls GreCreateJournalFile, the hardcopy driver processes the calls to produce the first band while the journal file is being accumulated. Otherwise, the COM_DRAW command bit flag is turned *off* and the hardcopy driver is, in effect, told not to draw while the journal file is being accumulated.

When the journal file is complete, if the JNL_DRAW_OPTIMIZATION bit flag was set *on* when the function GreCreateJournalFile was called, the hardcopy driver passes the completed band to the spooler or hardcopy device. It then plays the journal file and reprocesses the calls to produce the next band. Otherwise, the hardcopy driver plays the journal file and reprocesses the calls to produce all bands including the first.

Graphics Engine Functions by Category

Related graphics engine functions can be grouped together into the following categories:

Device Context Functions

- GreCloseDC (see page 11-4)
- GreGetHandle (see page 11-29)
- GreGetProcessControl (see page 11-30)
- GreOpenDC (see page 11-33)
- GreQueryEngineVersion (see page 11-42)
- GreResetDC (see page 11-50)
- GreRestoreDC (see page 11-52)
- GreSaveDC (see page 11-54)
- GreSetHandle (see page 11-67)
- GreSetProcessControl (see page 11-68)

Device Support Functions

- GreCreateBitmap (see page 11-7)
- GreDeleteBitmap (see page 11-17)
- GreGetAttributes (see page 11-21)
- GreGetBitmapDimension (see page 11-22)
- GreGetBitmapParameters (see page 11-23)
- GreGetDefaultArcParameters (see page 11-26)
- GreGetDefaultAttributes (see page 11-27)
- GreGetDefaultViewingLimits (see page 11-28)
- GreInitializeAttributes (see page 11-31)
- GreSelectBitmap (see page 11-56)
- GreSetAttributes (see page 11-58)
- GreSetBitmapDimension (see page 11-60)
- GreSetDefaultArcParameters (see page 11-62)
- GreSetDefaultAttributes (see page 11-63)
- GreSetDefaultViewingLimits (see page 11-65)
- GreSetGlobalAttribute (see page 11-66)

Font Functions

- GreCreateLogicalFont (see page 11-14)
- GreLoadFont (see page 11-32)
- GreQueryCodePageVector (see page 11-41)
- GreQueryFontAttributes (see page 11-43)
- GreQueryFontFileDescriptions (see page 11-44)
- GreQueryFonts (see page 11-45)
- GreQueryLogicalFont (see page 11-46)
- GreUnLoadFont (see page 11-46)

Journaling Functions (Hardcopy drivers only)

- GreCreateJournalFile (see page 11-12)
- GreDeleteJournalFile (see page 11-18)
- GreOpenJournalFile (see page 11-37)
- GrePlayJournalFile (see page 11-38)
- GreStartJournalFile (see page 11-69)
- GreStopJournalFile (see page 11-71)

LCID Functions

- GreCopyDCLoadData (see page 11-5)
- GreQueryBitmapHandle (see page 11-40)
- GreSetBitmapID (see page 11-61)

Set ID Functions

- GreDeleteSetId (see page 11-20)
- GreQueryNumberSetIds (see page 11-47)
- GreQuerySetIds (see page 11-48)

GreCloseDC

```
#define INCL_GRE_DCS

BOOL GreCloseDC (hdc, pInstance, lFunction)
```

This function closes a device context. If the device context is a memory DC, which has a bit map selected, the engine deselects the bit map before destroying the DC. This function deletes any visible regions and clip regions selected into the DC.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCloseDC

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_CODEPAGE
PMERR_INV_COORDINATE
PMERR_INV_DC_TYPE
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SCAN_START

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCopyDCLoadData

```
#define INCL_GRE_LCID

BOOL GreCopyDCLoadData (hdc, flCmd, hdcSrc, pInstance, lFunction)
```

This function copies the loaded fonts, bit maps, color table, and default attributes from one DC to another. Device-dependent attributes such as pick aperture, character cell, and marker cell are copied only when they have been set by the application. Otherwise, the target device defaults are used.

In response to this call, the graphics engine does the following:

- Transfers the contents of the source DC's lcid table to the target DC's lcid table.
- Translates any bit maps when the devices associated with the source and target DCs are different.
- Calls GreCreateLogicalFont for the target DC, as necessary.
- Transfers the color table and checks that the resulting color table in the target DC is the same as the one in the source DC.
- Resets the source DC using GreResetDC. When the save level of the source DC is not *1*, it is restored to the default value before GreResetDC is called. The color table is also reset to the default value.

The values of the GPI handles in the source DC are preserved. This function fails if the target DC lcid table already has lcids set in the range specified. If the target DC does not support LCOL_REALIZABLE (see "GreCreateLogColorTable" on page 8-34), a warning (PMERR_REALIZE_NOT_SUPPORTED) is raised and the color table is treated as nonrealizable.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Handle of target DC.
flCmd	ULONG	Flags indicating which fonts to copy. See below.
hdcSrc	HDC	Handle of source DC.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCopyDCLoadData.

flCmd Valid flags are:

LCID_RANGE_GPI	GPI
LCID_RANGE_AVIO	AVIO
LCID_RANGE_BOTH	GPI and AVIO.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Remarks: This function has the following affect on fonts:

- Any logical font in force on the source DC is reloaded onto the target with the original parameters specified by the application.
- A generic font selected by Match ID can be selected into the target DC provided that it is suitable. For example, an image font cannot be selected for a hardcopy DC.
- Any device font (selected by Match ID) can be selected for the target DC provided that the new device has a font with the same Match ID. If not, the defaults are used. It is the responsibility of the presentation driver to manage this. When it is necessary to use a specific font on the new device, the presentation driver should call GreQueryFonts to determine the characteristics and Match ID of the font for the new device and then to reload it.
- A warning is raised when it is not possible to reload a font on association (not on reassociation). This warning is one of the following:

  PMERR_FONT_NOT_LOADED
  PMERR_KERNING_NOT_SUPPORTED.

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCreateBitmap

```
#define INCL_GRE_DEVSUPPORT

HBITMAP GreCreateBitmap (hdc, pInfoHd, flUsage, pBitmap, paInfo, pInstance, lFunction)
```

This function creates a bit map and returns its handle.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
pInfoHd	PBITMAPINFOHEADER	Pointer to BITMAPINFOHEADER or BITMAPINFOHEADER2 structure for new bit map. See below.
flUsage	ULONG	Additional information, used when creating a new bit map. See below.
pBitmap	PBYTE	Pointer to bit-map initialization data.
paInfo	PBITMAPINFO	Pointer to BITMAPINFO2 or BITMAPINFO2 structure for initialization data. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCreateBitmap.

pInfoHd Pointer to either a BITMAPINFOHEADER structure:

cbFix Length in bytes of this structure
cx Bit-map width
cy Bit-map height
cPlanes Number of color planes, *1* if standard format
cBitCount Number of adjacent color bits per pel

Each plane has ((cx*cBitCount+31)/32*4*cy) bytes.

Or pointer to a BITMAPINFOHEADER2 structure:

cbFix Length in bytes of this structure

cx Bit-map width

cy Bit-map height

cPlanes Number of color planes, *1* if standard format

cBitCount Number of adjacent color bits per pel

ulCompression Compression scheme used to store the bit map:

BCA_UNCOMP Bit map is uncompressed (the only valid value).

cbImage Length of bit-map storage data in bytes. If the bit map is uncompressed, *0* (default) can be specified for this.

cxResolution Horizontal component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution Vertical component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cclrUsed The number of color indexes from the color table that are used by the bit map. If it is *0* (default), all the indexes are used. If it is non-zero, only the first cclrUsed entries in the table are accessed by the system; further entries can be omitted.

For standard formats with a cBitCount of *1*, *4*, or *8* (and cPlanes=*1*), any indexes beyond cclrUsed are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, cclrUsed is the number of colors used by the bit map.

cclrImportant Minimum number of colors indexes for satisfactory appearance of the bit map. More colors can be used in the bit map. However, it is not necessary to assign them to the device palette. These additional colors can be mapped to the nearest colors available. Zero (default) means that all entries are important. For a 24-bitcount standard format, the cclrImportant colors are also listed in the color table relating to this bit map.

usUnits Units of measure of the horizontal and vertical components of resolution:

BRU_METRIC (Default.) Pels per meter.

usReserved Reserved field. If present, it must be *0*.

usRecording Recording algorithm, the format in which bit-map data is recorded:

BRA_BOTTOMUP (Default.) Scan lines are recorded from bottom-to-top.

usRendering Half-toning algorithm used to record bit-map data that has been digitally half-toned:

BRH_NOTHALFTONED (Default.) Bit-map data not half-toned.

BRH_ERRORDIFFUSION Error diffusion or damped error diffusion algorithm

BRH_PANDA Processing algorithm for non-coded document acquisition

BRH_SUPERCIRCLE Super circle algorithm

cSize1 Size value *1*. If BRH_ERRORDIFFUSION is specified in usRendering, cSize1 is the error damping as a percentage in the range *0–100*. A value of *100%* indicates no damping. A value of *0%* indicates that any errors are not diffused.

If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize1 is the x-dimension of the pattern used in pels.

cSize2 Size value 2. If BRH_ERRORDIFFUSION is specified in usRendering, this parameter is ignored. If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize2 is the y-dimension of the pattern used in pels.

ulColorEncoding Color encoding:

BCE_RGB (Default.) Each element in the color array is an RGB2 data type.

ulIdentifier Reserved for applications.

flUsage The only defined flag is:

CBM_INIT When set, the pBitmap and paInfo parameters are used to initialize the newly created bit map. If this flag is not set, the bit map initialization is device dependent. It is assumed that sufficient data is supplied to initialize the whole bit map.

Other flags (16−31) can be used for special purposes known to be supported by a particular presentation driver.

paInfo Pointer to either a BITMAPINFO structure:

cbFix Length of structure
cx Bit-map width
cy Bit-map height
cPlanes Number of color planes, *1* if standard format
cBitCount Number of adjacent color bits per pel
argbColor[ ] Color table array of RGB structures:
bBlue
bGreen
bRed.

Or to a BITMAPINFO2 structure:

cbFix Length in bytes of this structure

cx Bit-map width

cy Bit-map height

cPlanes Number of color planes, *1* if standard format

cBitCount Number of adjacent color bits per pel

ulCompression Compression scheme used to store the bit map:

BCA_UNCOMP Bit map is uncompressed (the only valid value).

cbImage Length of bit-map storage data in bytes. If the bit map is uncompressed, *0* (default) can be specified for this.

cxResolution Horizontal component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution Vertical component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cclrUsed The number of color indexes from the color table that are used by the bit map. If it is *0* (default), all the indexes are used. If it is non-zero, only the first cclrUsed entries in the table are accessed by the system; further entries can be omitted.

For standard formats with a cBitCount of *1*, *4*, or *8* (and cPlanes=*1*), any indexes beyond cclrUsed are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, cclrUsed is the number of colors used by the bit map.

cclrImportant Minimum number of colors indexes for satisfactory appearance of the bit map. More colors can be used in the bit map but it is not necessary to assign them to the device palette. These additional colors can be mapped to the nearest colors available. Zero (default) means that all entries are important. For a 24-bitcount standard format, the cclrImportant colors are also listed in the color table relating to this bit map.

usUnits Units of measure of the horizontal and vertical components of resolution:

BRU_METRIC (Default.) Pels per meter.

usReserved Reserved field. If present, it must be *0*

usRecording Recording algorithm, the format in which bit-map data is recorded:

BRA_BOTTOMUP (Default.) Scan lines are recorded from bottom-to-top.

usRendering Half-toning algorithm used to record bit-map data that has been digitally half-toned:

BRH_NOTHALFTONED (Default.) Bit-map data not half-toned.

BRH_ERRORDIFFUSION Error diffusion or damped error diffusion algorithm

BRH_PANDA Processing algorithm for non-coded document acquisition

BRH_SUPERCIRCLE Super circle algorithm

cSize1 Size value *1*. If BRH_ERRORDIFFUSION is specified in usRendering, cSize1 is the error damping as a percentage in the range *0—100*. A value of *100%* indicates no damping a value of *0%* indicates that any errors are not diffused.

If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize1 is the x-dimension of the pattern used in pels.

cSize2 Size value *2*. If BRH_ERRORDIFFUSION is specified in usRendering, this parameter is ignored. If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize2 is the y-dimension of the pattern used in pels.

ulColorEncoding Color encoding:

BCE_RGB (Default.) Each element in the color array is an RGB2 data type.

ulIdentifier Reserved for application use.

argbColor[ ] Color table array of RGB2 structures:

bBlue
bGreen
bRed
fcOptions Reserved. Must be *0*.

Note: The same bit-map file format is used for bit maps, icons and pointers. For details, refer to the *OS/2 2.0 Presentation Manager Programming Reference*.

Return Codes: On completion, the handling routine must return the handle of the bit map (hbm), or *0* if an error occurs.

Possible Errors Detected: Error codes posted by the engine for this function include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_HDC
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SCAN_START
PMERR_INV_USAGE_PARM

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: Bit-map size is limited by available memory; the maximum width and height are 64KB. The following standard bit-maps formats should normally be used:

Bitcount	Planes
1	1
4	1
8	1
24	1

Display drivers must support at least 4 bits per pel. For other devices, the presentation driver must be able to create and accept any of the standard formats even though they might not use the color information.

The DC handle supplied to this function must never be NULL. Bit maps always belong to some device. The bit map is created on the device specified and can be selected to a different device later as the engine can handle the transfer of bits from one device to another. When the value specified for cPlanes or cBitcount is incompatible with the physical device specified by the DC handle, an error is raised.

GreCreateJournalFile

```
#define INCL_GRE_JOURNALING

ULONG GreCreateJournalFile (pszFileName, flOption, cSize, pInstance, lFunction)
```

This function creates a journal file on disk. The presentation driver calls GreCreateJournal when responding to DevEscape DEVESC_STARTDOC.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
pszFileName	PSZ	Pointer to a string containing the file name.
flOption	ULONG	Options. See below.
cSize	ULONG	Size. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCreateJournalFile.

flOptions Defined values are:

JNL_TEMP_FILE An ordinary temporary journal file is created. pszFileName is ignored.

JNL_PERM_FILE A permanent journal file is created. pszFileName points to a fully qualified path or file name.

JNL_ENGINERAM_FILE A memory journal file is created in shared memory allocated by the engine. pszFileName is ignored.

JNL_USERRAM_FILE A memory journal file is created in memory supplied by the caller. The location in memory is identified by the pointer passed in pszFileName.

JNL_DRAW_OPTIMIZATION If set, the process control flag PCTL_DRAW is reset (optimization occurs). The Draw bit is not affected.

JNL_BOUNDS_OPTIMIZATION If set, the process control flag PCTL_BOUND is reset (BOUNDS is turned *off*). Otherwise, current behavior continues.

cSize If greater than *0*, cSize is an indication as to how large the file must be. If flOption is JNL_USERRAM_FILE, cSize must be greater than *0* and is the size of the buffer, which cannot be extended.

If cSize is *0*, the calling routine does not know the size of the file.

Return Codes: This function returns the journal file handle (ULONG), or if an error occurs, NULL.

Possible Errors Detected: If this function fails, the graphics engine will set one of the following error codes:

PMERR_BASE_ERROR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_JOURNAL_OPTION
PMERR_RAM_JNL_FILE_TOO_SMALL

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreCreateLogicalFont

```
#define INCL_GRE_FONTS

LONG GreCreateLogicalFont (hdc, lcid, pchName, pAttrs, pInstance, lFunction)
```

This function sets the local identifier (lcid) for a logical font. The parameters passed to the function identify the name and attributes of the required font. The graphics engine selects a font from the list of available fonts that provides the best match for the font attributes addressed by pAttrs.

The selection is made in one of two ways:

1. If the lMatch attribute is non-zero, the calling program has already determined (from a call to GreQueryFonts) which font it requires. In this case, the graphics engine selects the font identified by the lMatch and szFaceName attributes (or, if szFaceName is not specified, lMatch alone).
2. If lMatch is *0* or a suitable match could not be found, the system examines the other fields in the attributes structure and selects the font that gives the best match.

If no match is found, the default font is used. Once assigned, the system will not change the relationship of a specific lcid to a specific font.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
lcid	LONG	Local identifier that is to be assigned to the font.
pchName	PSTR8	Pointer to an eight-character name used to describe the logical. font
pAttrs	PFATTRS	Pointer to font attributes structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreCreateLogicalFont.

pAttrs Pointer to a FATTRS structure containing the font attributes:

usRecordLength Length in bytes of this structure

fsSelection Flags that can be set to define those features to be simulated by the system:

FATTR_SEL_ITALIC	Italic characters are required.
FATTR_SEL_UNDERSCORE	Underscored characters are required.
FATTR_SEL_STRIKEOUT	Strikeout characters are required.
FATTR_SEL_BOLD	Bold characters are required.

lMatch The match number for the font. GreQueryFonts returns a unique match number for each font that, together with szFaceName, can be used for font selection. If the match number is negative, a device font is selected. If it is positive, an engine font is selected.

szFaceName The typeface name to which the font is designed. If the szFaceName is provided, an attempt is made to select the font with that face name. If the szFaceName is blank (0 length or NULL), then a default font is used.

idRegistry The Registry number for the font. This should be set to the value returned by GreQueryFonts.

usCodePage Defines the code page supported by the font.

lMaxBaselineExt The required (average) height above the baseline for uppercase characters in world coordinates. For outline and transformable fonts (FATTR_FONTUSE_OUTLINE and FATTR_FONTUSE_TRANSFORM), lMaxBaselineExt should be set to *0*.

lAveCharWidth For image fonts, the required average inter-character increment for the font in world coordinates. For outline and transformable fonts (FATTR_FONTUSE_OUTLINE and FATTR_FONTUSE_TRANSFORM), lAveCharWidth should be set to *0*.

fsType Type indicator. Setting fsType to FATTR_TYPE_KERNING indicates that kerning is to be used if the font provides kerning information.

fsFontUse Flags containing information about how the font is to be related to the character attributes:

FATTR_FONTUSE_NOMIX
Specifies that permissive mixing is allowed when the font is used. The engine can ignore any interaction with graphics primitives and can use *overpaint* and *leave alone* as the foreground and background mixes rather than using the current mix attributes.

FATTR_FONTUSE_OUTLINE
Specifies that the font must be an outline font. When the font is not defined by lMatch and FATTR_FONTUSE_OUTLINE is specified, the system searches for an outline font. If the search fails, a default font is selected. When the font is not defined by lMatch and FATTR_FONTUSE_OUTLINE is not set, the system searches for an image font that matches lMaxBaselineExt and lAveCharWidth. If this fails, it searches for an outline font with the required szFaceName and fsSelection flags.

FATTR_FONTUSE_TRANSFORMABLE
Specifies that the font must be transformable (that is, it can be scaled, rotated and sheared). Transformable fonts are used only in CM_MODE3. Non-transformable fonts can be used in CM_MODE1 or CM_MODE2 but not in CM_MODE3.

Return Codes: This function returns a LONG value indicating whether the font has been matched:

FONT_DEFAULT The font was not matched. The default font is to be used.
FONT_MATCH The font has been matched successfully.
GPI_ERROR Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_INV_COORD_SPACE
PMERR_INV_EXTENDED_LCID
PMERR_INV_FONT_ATTRS
PMERR_INV_FONTDEF
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SETID
PMERR_SETID_IN_USE

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: The interaction between fonts and character attributes depends on the state of the FATTRS_FONTUSE_TRANSFORMABLE flag in the font attributes structure. When this flag is set:

- The size of the characters is determined by the values of the character attributes at the time that the characters are drawn.
- The characters are positioned, rotated, and sheared, as required.
- No checking is done.
- Any transformation is performed by mapping the box defined by the FONTMETRICS parameters, xMaxCharInc and yEmHeight, to the character box under the influence of character angle and shear.

When the FATTRS_FONTUSE_TRANSFORMABLE flag is not set, the lAveCharWidth and lBaselineExt parameters in the font attributes structure define the size of the font to be used. The character box attribute has no effect.

Transformable fonts cannot be used in Character Modes 1 and 2. Non-transformable fonts cannot be used in Character Mode 3. If the font is not compatible with the character mode, the engine raises an error when the presentation driver attempts to draw characters. The characteristics of the character modes are:

CM_MODE1 The position of characters after the first character is determined by the font metrics information. Character box, angle, shear, extra, break extra, and spacing are ignored.

CM_MODE2 The position of characters is determined by the font metrics information and the character attributes. Characters are not scaled, rotated, or sheared.

CM_MODE3 The position of characters is determined by the font metrics information and all character attributes. Characters can be scaled, rotated, and sheared.

Positioning is performed by using the character reference point defined within the font. When characters that are not hollow are drawn using an outline font, they are filled using the character foreground color and mix.

GreDeleteBitmap

```
#define INCL_GRE_DEVSUPPORT

BOOL GreDeleteBitmap (hbm, pInstance, lFunction)
```

This function deletes the bit map identified by hbm. If the bit map is currently selected into a DC, GreDeleteBitmap returns an error.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hbm	HBITMAP	Bit-map handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeleteBitmap

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BITMAP_IS_SELECTED
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HBITMAP_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SCAN_START

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeleteJournalFile

```
#define INCL_GRE_JOURNALING

BOOL GreDeleteJournalFile (hJournal, pInstance, lFunction)
```

This function deletes a journal file and frees any objects associated with the journal file handle (such as compatible DCs, private clone regions or bit maps, and global memory segments). When the handle belongs to a temporary file, the file is also deleted. Finally, the file handle itself is freed.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hJournal	ULONG	Journal file handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeleteJournalFile

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_DC_TYPE
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_HJOURNAL

PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_END_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_INV_METAFILE
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SCAN_START
PMERR_INV_SETID
PMERR_JFILE_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_UNSUPPORTED_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreDeleteSetId

```
#define INCL_GRE_SETID

BOOL GreDeleteSetId (hdc, lcid, pInstance, lFunction)
```

This function deletes the character set, marker set, or pattern set identified by lcid. Base sets cannot be deleted. An error is logged if GreDeleteSetId is called to delete the current character, marker, or pattern set.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
lcid	LONG	Local identifier. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreDeleteSetId.

lcid When identifying a bit map, only the lcid is deleted. The bit map will have no LCID but it will still exist. When lcid = LCID_ALL, all loaded graphics local identifiers such as logical fonts and Bit-Map IDs are destroyed. In this case, AVIO fonts are unaffected and can only be deleted explicitly.

LCID_AVIO_1, LCID_AVIO_2, and LCID_AVIO_3 represent AVIO fonts 1, 2, and 3, respectively.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_INV_COORD_SPACE
PMERR_INV_EXTENDED_LCID
PMERR_INV_FONTDEF
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SETID
PMERR_SETID_IN_USE
PMERR_SETID_NOT_FOUND

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetAttributes

```
#define INCL_GRE_DEVSUPPORT

LONG GreGetAttributes (hdc, lPrimType, flAttrsMask, pAttrs, pInstance, lFunction)
```

This function returns the current value of the attributes indicated in flAttrsMask. When a specified attribute is currently set to its default value, the corresponding flag in the returned defaults mask is set and the returned value for this attribute is undefined.

The graphics engine either:

- Returns the value of each specified attribute and resets the corresponding bit in the returned mask, or
- Sets the bit in the returned mask to indicate that the specified attribute is set to its default. Notice that the the corresponding value in the attribute buffer is *not* valid and is likely to have been overwritten by the engine.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
lPrimType	LONG	Bundle primitive type. See below.
flAttrsMask	ULONG	Attribute mask. See below.
pAttrs	PBUNDLE	Pointer to the fixed format bundle record containing the attributes returned.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetAttributes.

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE	Line attribute bundle
PRIM_CHAR	Character attribute bundle
PRIM_MARKER	Marker attribute bundle
PRIM_AREA	Pattern attribute bundle
PRIM_IMAGE	Image attribute bundle.

flAttrsMask Specifies the attributes to be returned. This mask contains a bit corresponding to each attribute in the bundle record. For each set bit, the graphics engine returns the corresponding attribute values and default mask bits.

pAttrs The returned attribute value (bundle). The only fields that are updated are those whose corresponding flags in flAttrsMask have been set.

Return Codes: This function returns the default attribute bit mask. Only bits with corresponding set bits in flAttrsMask are updated. Otherwise, this function returns the error, GPI_ALTERROR.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

GreGetBitmapDimension

```
#define INCL_GRE_DEVSUPPORT

BOOL GreGetBitmapDimension (hbm, pDimension, pInstance, lFunction)
```

This function renders height and width values for the bit map indicated by hbm. These are values that have been set by a previous call to GreSetBitmapDimension. They are not used by the system.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hbm	HBITMAP	Bit-map handle
pDimension	PSIZEL	Pointer to width and height values in 0.1mm units
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetBitmapDimension

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_HBITMAP

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetBitmapParameters

```
#define INCL_GRE_DEVSUPPORT

BOOL GreGetBitmapParameters (hbm, pInfoHd, pInstance, lFunction)
```

This function returns, in the buffer addressed by pInfoHd, header information for the specified bit map. The header information is returned as a BITMAPINFOHEADER or BITMAPINFOHEADER2 structure and gives details such as the width, height, number of planes, and number of bits per pel.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hbm	HBITMAP	Bit-map handle.
pInfoHd	PBITMAPINFOHEADER	Pointer to a BITMAPINFOHEADER or BITMAPINFOHEADER2 structure where the returned information is stored. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetBitmapParameters.

pInfoHd Pointer to either a BITMAPINFOHEADER structure:

cbFix Length in bytes of this structure
cx Bit-map width
cy Bit-map height
cPlanes Number of color planes, *1* if standard format
cBitCount Number of adjacent color bits per pel

Each plane has ((cx*cBitCount+31)/32*4*cy) bytes.

Or pointer to a BITMAPINFOHEADER2 structure:

cbFix Length in bytes of this structure

cx Bit-map width

cy Bit-map height

cPlanes Number of color planes, *1* if standard format

cBitCount Number of adjacent color bits per pel

ulCompression Compression scheme used to store the bit map:

BCA_UNCOMP Bit map is uncompressed (the only valid value).

cbImage Length of bit map storage data in bytes. If the bit map is uncompressed, *0* (default) can be specified for this.

cxResolution Horizontal component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cyResolution Vertical component of the resolution of the target device. That is, the resolution of the device the bit map is intended for in the units specified by usUnits. This information enables an application to select from a resource group the bit map that best matches the characteristics of the current output device.

cclrUsed The number of color indexes from the color table that are used by the bit map. If it is *0* (default), all the indexes are used. If it is non-zero, only the first cclrUsed entries in the table are accessed by the system; further entries can be omitted.

For standard formats with a cBitCount of *1*, *4*, or *8* (and cPlanes=*1*), any indexes beyond cclrUsed are not valid. For example, a bit map with 64 colors can use the 8-bitcount format without having to supply the other 192 entries in the color table. For the 24-bitcount standard format, cclrUsed is the number of colors used by the bit map.

cclrImportant Minimum number of colors indexes for satisfactory appearance of the bit map. More colors can be used in the bit map. However, it is not necessary to assign them to the device palette. These additional colors can be mapped to the nearest colors available. Zero (default) means that all entries are important. For a 24-bitcount standard format, the cclrImportant colors are also listed in the color table relating to this bit map.

usUnits Units of measure of the horizontal and vertical components of resolution:

BRU_METRIC (Default.) Pels per meter.

usReserved Reserved field. If present, it must be *0*.

usRecording Recording algorithm, the format in which bit-map data is recorded:

BRA_BOTTOMUP (Default.) Scan lines are recorded from bottom-to-top.

usRendering Half-toning algorithm used to record bit-map data that has been digitally half-toned:

BRH_NOTHALFTONED (Default.) Bit-map data not half-toned.

BRH_ERRORDIFFUSION Error diffusion or damped error diffusion algorithm

BRH_PANDA Processing algorithm for non-coded document acquisition

BRH_SUPERCIRCLE Super circle algorithm

cSize1 Size value *1*. If BRH_ERRORDIFFUSION is specified in usRendering, cSize1 is the error damping as a percentage in the range *0–100*. A value of *100%* indicates no damping. A value of *0%* indicates that any errors are not diffused.

If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize1 is the x-dimension of the pattern used in pels.

cSize2 Size value 2. If BRH_ERRORDIFFUSION is specified in usRendering, this parameter is ignored. If the BRH_PANDA or BRH_SUPERCIRCLE is specified, cSize2 is the y-dimension of the pattern used in pels.

ulColorEncoding Color encoding:

BCE_RGB (Default.) Each element in the color array is an RGB2 data type.

ulIdentifier Reserved for application use

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_HBITMAP

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetDefaultArcParameters

```
#define INCL_GRE_DEVSUPPORT

BOOL GreGetDefaultArcParameters (hdc, pArcParms, pInstance, lFunction)
```

This function stores the default arc parameters in the buffer addressed by pArcParms.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
pArcParms	PARCPARAMS	Pointer to ARCPARAMS structure
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetDefaultArcParameters

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetDefaultAttributes

```
#define INCL_GRE_DEVSUPPORT

BOOL GreGetDefaultAttributes (hdc, lPrimType, flAttrsMask, pAttrs, pInstance, lFunction)
```

This function returns the default values of the attributes indicated in flAttrsMask.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
lPrimType	LONG	Bundle primitive type. See below.
flAttrsMask	ULONG	Attribute mask. See below.
pAttrs	PBUNDLE	Pointer to the fixed format bundle record containing the attributes returned.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetDefaultAttributes.

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE	Line attribute bundle
PRIM_CHAR	Character attribute bundle
PRIM_MARKER	Marker attribute bundle
PRIM_AREA	Pattern attribute bundle
PRIM_IMAGE	Image attribute bundle.

flAttrsMask Specifies the attributes to be returned. This mask contains a bit corresponding to each attribute in the bundle record. For each set bit, the engine returns the corresponding attribute values and default mask bits.

Return Codes: This function returns the default attribute bit mask. Only bits with corresponding set bits in flAttrsMask are updated. Otherwise, this function returns the error, ATTRS_ERROR.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetDefaultViewingLimits

```
#define INCL_GRE_DEVSUPPORT

BOOL GreGetDefaultViewingLimits (hdc, prclViewingLimits, pInstance, lFunction)
```

This function loads the array indicated by prclViewingLimits with the default boundaries of the viewing window in graphic model space coordinates.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
prclViewingLimits	PRECTL	Pointer to limits of viewing area. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetDefaultViewingLimits.

prclViewingLimits RECTL structure:

xLeft Minimum x-coordinate of viewing limits
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of viewing limits
yTop Maximum y-coordinate.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetHandle

```
#define INCL_GRE_DCS

LONG GreGetHandle (hdc, iIndex, pInstance, lFunction)
```

This function returns the handle or variable (stored in the DC corresponding to iIndex) previously set by GreSetHandle.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
iIndex	ULONG	Index value of the returned handle in the range *0*—*3*. A value of *1* can be used to get the associated AVIO presentation space handle.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetHandle.

Return Codes: On completion, the graphics engine returns the handle requested (hHandle), or GPI_ALTERROR if an error occurs.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreGetProcessControl

```
#define INCL_GRE_DCS

LONG GreGetProcessControl (hdc, pInstance, lFunction)
```

This function returns the process control flags.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreGetProcessControl

Return Codes: This function returns the process control flags (flProcess), or DCTL_ERROR if an error occurs. The flags returned are:

PCTL_DRAW	Draw flag
PCTL_BOUND	GPI_BOUNDS flag
PCTL_CORRELATE	Correlate flag
PCTL_USERBOUNDS	USER_BOUNDS flag
PCTL_AREA	When set, an area definition is in progress.
COM_PATH	When set, a path definition is in progress.

Other flags are reserved and should be ignored.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreInitializeAttributes

```
#define INCL_GRE_DEVSUPPORT

BOOL GreInitializeAttributes (hdc, flOptions, pInstance, lFunction)
```

This function sets the current default attributes to the initial standard default values. It can also be used to reset the current attributes to the current default values. GreInitializeAttributes affects all bundle attributes that can be set with GreSetAttributes, arc parameters, and viewing limits.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	Option flags. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreInitializeAttributes.

flOptions Valid flags are:

INAT_DEFAULTATTRIBUTES When set, all the current default attributes are set to their initial standard default values.

INAT_CURRENTATTRIBUTES When set, all the current attributes are set to their current default values.

Notice that when both flags are set, INAT_DEFAULTATTRIBUTES is processed first.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

GreLoadFont

```
#define INCL_GRE_FONTS

BOOL GreLoadFont (pszFilename, pInstance, lFunction)
```

This function loads fonts from a resource file. All fonts in the file are private and are only available to the process that called this function.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
pszFilename	PSZ	Pointer to a NULL-terminated string containing path and name of the font file, identified by the file extension FON
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreLoadFont

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_FONT_FILE_DATA

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreOpenDC

```
#define INCL_GRE_DCS

HDC GreOpenDC (hdc, ulType, pszToken, cData, pszData, pInstance, lFunction)
```

This function creates an output device context (DC). The new device context inherits the current code page of the process that created it. Subsequent calls to DosSetCp do not alter the code page of an existing DC. Default VIO and KBD code pages are always in the last code page set by any application process.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle. See below.
ulType	ULONG	Type of DC being opened. See below.
pszToken	PSZ	This parameter is ignored by the graphics engine.
cData	LONG	Number of elements in the data structure.
pszData	PDEVOPENDATA	Pointer to data structure. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreOpenDC.

hdc When ulType is OD_MEMORY, this parameter is a handle to a device context compatible with bit maps that are used with this device context. If this is NULL, compatibility with the screen is assumed.

ulType Type of device context:

- **OD_QUEUED** The DC is for a hardcopy device for which output is to be queued by the spooler.
- **OD_DIRECT** The DC is for a device for which output is not to be queued by the spooler.
- **OD_INFO** This is similar to OD_DIRECT except that it is only used to retrieve information such as font metrics. Drawing can be done to a presentation space associated with such a DC but no output medium is updated.
- **OD_MEMORY** The new DC is a memory DC used to contain a bit map. hdc identifies the device with which the bit map is to be compatible.

pszData Pointer to DEVOPENDATA structure:

- **pszLogAddress** Pointer to logical address, for example, *LPT1Q*.
- **pszDriverName** Pointer to presentation driver name, for example, *LASERJET*.
- **pdriv** Pointer to a DRIVDATA structure:
  - **cb** Size in bytes of this structure
  - **lVersion** Version number of the data. Version numbers are defined by the presentation driver.
  - **szDeviceName[32]** String identifying the device. Valid values are supplied by the presentation driver.

abGeneralData General data defined by the presentation driver. This does not contain pointers as they might not be valid when passed to the driver.

pszDataType Pointer to a data type of the queued file. Supported data types are:

PM_Q_STD
PM_Q_RAW

User-defined data types can also be supported.

pszComment Pointer to a description of the file that can be displayed by the spooler to the user

pszQueueProcName Pointer to name of queue processor

pszQueueProcParams Pointer to a string of queue processor parameters

pszSpoolerParams Pointer to a string of spooler parameters separated by one or more blanks. Valid parameters are:

FORM = aaa Identifies the form name for a print job. Multiple names are separated by commas (aaa,bbb,ccc). If this parameter is not present, the current form is used.

Form names are defined by the presentation driver. Valid names are those that would be returned from a call to the driver's GreQueryHardcopyCaps handling routine.

PRTY = n Identifies the priority for the print job. The priority can be any value from *1 – 99* (*1* is lowest priority). If this parameter is not present, the priority value defaults to *50*.

pszNetworkParams Pointer to a string of networking parameters, which are used only in a network environment. Their nature is defined by the network application.

Return Codes: On completion, the graphics engine returns the handle of the new device context (hdc), or DEV_ERROR if an error occurs.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COORD_SPACE

PMERR_INV_COORDINATE
PMERR_INV_DC_DATA
PMERR_INV_DC_TYPE
PMERR_INV_DRIVER_NAME
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SCAN_START

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When this function is called for the first time, the graphics engine performs the following sequence:

```
DosLoadModule("NEWDRIVER", &hDriver)
                /*****************************************************/
                /* Load the presentation driver DLL file.            */
                /*****************************************************/

DosGetProcAddr(hDriver, "OS2_PM_DRV_ENABLE", &Enable)
                /*****************************************************/
                /* Find the presentation driver's Enable function.   */
                /* See "OS2_PM_DRV_ENABLE."                          */
                /*****************************************************/

*Enable(FillLogicalDeviceBlock, &DispatchTable)
                /*****************************************************/
                /* The presentation driver must:                     */
                /*   Save the addresses of the engine simulations    */
                /*   Overwrite the dispatch table, as necessary      */
                /*   Hook the ExitList for the calling process       */
                /*****************************************************/

hPhysDev=*Enable(FillPhysicalDeviceBlock, pDevOpenStructure)
                /*****************************************************/
                /* Create the physical device block.                 */
                /*****************************************************/

pInstance=*Enable(EnableDeviceContext, hdc, Type, hPhysDev)
                /*****************************************************/
                /* The presentation driver must create an instance   */
                /* data structure for the DC.                        */
                /*****************************************************/

*Enable(CompleteOpenDC, hdc, pInstance)
                /*****************************************************/
                /* The presentation driver must inform the system    */
                /* that the DC is open and ready to receive output.  */
                /*****************************************************/
```

For subsequent calls to GreOpenDC for the same DC, the graphics engine calls Enable (EnableDeviceContext) to create a new instance for the DC and then calls Enable (CompleteOpenDC). When GreOpenDC is called for the same DC by a different process, the graphics engine calls DosLoadModule to load the presentation driver. It then calls Enable (FillLogicalDeviceBlock), Enable (EnableDeviceContext), and Enable (CompleteOpenDC).

GreOpenJournalFile

```
#define INCL_GRE_JOURNALING

ULONG GreOpenJournalFile (pszFileName, flOption, cBufSize, pInstance, lFunction)
```

This function opens a journal file for play. The journal file must exist and must contain valid journaled calls.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
pszFileName	PSZ	Pointer a string containing the file name
flOption	ULONG	See below.
cBufSize	ULONG	Size of buffer required for the file
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreOpenJournalFile

flOption An option flag that identifies the type of journal file. Valid values are:

JNL_PERM_FILE A disk file with the name pszFileName, which was created by using GreCreateJournalFile (JNL_PERM_FILE).

Bit 3 A shared memory file pointed to by pszFileName and already filled with complete journal records. This journal file might have been created by using GreCreateJournalFile (JNL_USERRAM_FILE) and filled by the engine, or it might have been produced in some other way.

Return Codes: This function returns the journal file handle (ULONG), or if an error occurs, NULL.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_JOURNAL_OPTION
PMERR_RAM_JNL_FILE_TOO_SMALL

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GrePlayJournalFile

```
#define INCL_GRE_JOURNALING

BOOL GrePlayJournalFile (hdc, hJournal, pInstance, lFunction)
```

This function plays a journal file to the specified DC. The journal file is read into memory and each journaled call is played. Each journaled record is processed before playing to fix-up data pointers, and create *clone objects* that is, (regions or bit maps), if necessary, from the journaled data. It is assumed that any single journaled function and associated data fits in a 32KB buffer. If the journaled record contains region rectangles or bit-map bits, they are not considered in this restriction.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
hJournal	ULONG	Journal file handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGrePlayJournalFile

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_BITMAP_NOT_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_HRGN_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INCORRECT_DC_TYPE
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_DC_DATA
PMERR_INV_DC_TYPE

PMERR_INV_DRIVER_NAME
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_HJOURNAL
PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_METAFILE
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_REGION_MIX_MODE
PMERR_INV_SCAN_START
PMERR_INV_USAGE_PARM
PMERR_JFILE_BUSY
PMERR_RAM_JNL_FILE_TOO_SMALL
PMERR_REGION_IS_CLIP_REGION

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryBitmapHandle

```
#define INCL_GRE_LCID

HBITMAP GreQueryBitmapHandle (hdc, lLcid, pInstance, lFunction)
```

This function gets the bit-map handle for the specified local identifier (lcid). When lLcid does not reference a bit map, an error is raised by the graphics engine.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
lLcid	LONG	Local identifier for which the bit-map handle is required
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryBitmapHandle

Return Codes: This function returns the bit-map handle (hbm), or GPI_ERROR if an error occurs.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BITMAP_NOT_SELECTED
PMERR_HDC_BUSY
PMERR_INV_HDC
PMERR_INV_SETID

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryCodePageVector

```
#define INCL_GRE_FONTS

PUSHORT GreQueryCodePageVector (ulCodePage, pInstance, lFunction)
```

This function returns a pointer to a vector of 256 WORDs, which is the code point to the glyph mapping number.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
ulCodePage	ULONG	Code page number
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryCodePageVector

Return Codes: This function returns a pointer to the code page vector, or GPI_ERROR if an error occurs.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryEngineVersion

```
#define INCL_GRE_DCS

LONG GreQueryEngineVersion (pInstance, lFunction)
```

This function returns the version number of the graphics engine.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryEngineVersion

Return Codes: GreQueryEngineVersion returns the engine version number (lVersion), or GPI_ALTERROR if an error occurs. For example, `0X000000200L = 2.0`.

GreQueryFontAttributes

```
#define INCL_GRE_FONTS

BOOL GreQueryFontAttributes (hdc, cbMetrics, pfmMetrics, pInstance, lFunction)
```

This function returns the metrics of the current font at the location addressed by pfmMetrics.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
cbMetrics	ULONG	Size, in bytes, of the font metrics buffer
pfmMetrics	PFONTMETRICS	Pointer to font metric block where the information is to be returned
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryFontAttributes

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SETID

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryFontFileDescriptions

```
#define INCL_GRE_FONTS

ULONG GreQueryFontFileDescriptions (pszFileName, pcFonts, paszNames, pInstance, lFunction)
```

This function determines whether a file is a font file, and (if so) returns the family and face names for the fonts in the file. The names are returned at the location addressed by paszNames. Typically, the calling routine calls GreQueryFontFileDescriptions twice. The first call sets pcFonts to *0* and determines the number of fonts in the file. When sufficient storage is allocated for the names, GreQueryFontFileDescriptions is called again with pcFonts pointing to the font count.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
pszFileName	PSZ	Pointer to the file path and name of the font file.
pcFonts	PULONG	Pointer to a count of the maximum number of family and face name pairs to be returned. On completion, this is updated to the number of pairs actually returned.
paszNames	PSZ	Pointer to an array of 2×pcFonts 32-bit fields in which the family and face name pairs are returned. The family name is the first in each pair.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryFontFileDescriptions.

Return Codes: This function returns the number of fonts not returned, or GPI_ALTERROR if an error occurs.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_FONT_FILE_DATA

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryFonts

```
#define INCL_GRE_FONTS

LONG GreQueryFonts (hdc, flOptions, pszFaceName, paMetrics, cMetricLen, pcFontCount, pInstance, lFunction)
```

This function returns the details of fonts that match the face name addressed by pszFaceName.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	Flags indicating whether private fonts, public fonts, or both are required. See below.
pszFaceName	PSZ	Pointer to a string specifying the face name. When this is NULL, all faces are matched.
paMetrics	PFONTMETRICS	Pointer to an array of font element records to which the metrics of matching fonts are returned. See below.
cMetricLen	ULONG	Length in bytes of each metrics structure in the paMetrics array.
pcFontCount	PULONG	Pointer to cFontCount that specifies the number of fonts for which metrics are required. On return, this is updated to the number of fonts for which metrics were returned.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryFonts.

flOptions These flags can be used in combination:

QFF_PUBLIC Enumerate public fonts
QFF_PRIVATE Enumerate private fonts

paMetrics This is a pointer to an array of up to cFontCount font metric records, each of which contains a maximum of cMetricLen bytes. Notice that for multi-code page fonts the usCodePage field has no meaning and is set to *0*.

Return Codes: This function returns the number of fonts not returned, or GPI_ALTERROR if an error occurs.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_COORD_SPACE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_OR_INCOMPAT_OPTIONS

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryLogicalFont

```
#define INCL_GRE_FONTS

BOOL GreQueryLogicalFont (hdc, lcid, pchName, pLogFont, cLogFont, pInstance, lFunction)
```

This function returns the 8-character name and attributes for the logical font that is defined for the specified lcid. The data is returned in the locations addressed by pchName and pLogFont. When lcid identifies a Bit-Map ID, an error is raised by the graphics engine.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
lcid	LONG	Local identifier for the logical font
pchName	PSTR8	Pointer to an 8-character name used to describe the logical font
pLogFont	PFATTRS	Pointer to a font attribute structure
cLogFont	ULONG	Number of bytes of font attribute information requested
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryLogicalFont

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SETID

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQueryNumberSetIds

```
#define INCL_GRE_SETID

LONG GreQueryNumberSetIds (hdc, lRange, pInstance, lFunction)
```

This function returns the total number of LCIDs such as logical fonts and Bit-Map IDs that have been created.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
lRange	ULONG	Indicates whether GPI, or AVIO LCIDs, or both are to be returned. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQueryNumberSetIds.

lRange Valid ranges are:

LCID_RANGE_GPI	GPI
LCID_RANGE_AVIO	AVIO
LCID_RANGE_BOTH	GPI and AVIO.

Return Codes: This function returns the number of LCIDs, or GPI_ALTERROR if an error occurs.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC
PMERR_INV_SETID
PMERR_INV_SETID_TYPE

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreQuerySetIds

```
#define INCL_GRE_SETID

BOOL GreQuerySetIds (hdc, cSets, paTypes, paszNames, paLcid, lRange, pInstance, lFunction)
```

This function returns a list of created LCIDs with their names and types in the buffers addressed by paLcid, paszNames, and paTypes.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
cSets	LONG	Number of sets to return.
paTypes	PLONG	Pointer to an array of cSets elements indicating the type of the corresponding set. See below.
paszNames	PSTR8	Pointer to an array of 8-character names associated with the corresponding LCIDs. For a bit map, the name string is filled with *0s*.
paLcid	PLONG	Pointer to an array of cSets elements to which the lcids are returned.
lRange	ULONG	Indicates whether GPI, or AVIO local identifiers, or both are to be returned. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreQuerySetIds.

paTypes Each element in this array is a LONG value indicating the type of the corresponding local identifier:

LCIDT_FONT Logical font
LCIDT_BITMAP Bit-Map ID.

lRange Valid ranges are:

LCID_RANGE_GPI GPI
LCID_RANGE_AVIO AVIO
LCID_RANGE_BOTH GPI and AVIO.

LCID_AVIO_1, LCID_AVIO_2, and LCID_AVIO_3 represent AVIO sets of 1, 2, and 3.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_SETID
PMERR_INV_SETID_TYPE

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When cSets is greater than the number of lcids, the unused elements of paTypes and paLcid are set to *0*.

GreResetDC

```
#define INCL_GRE_DCS

BOOL GreResetDC (hdc, flOptions, pInstance, lFunction)
```

Calling this function restores a DC to its created state. All objects such as fonts, patterns and paths are deleted. All attributes are set to their defaults. Any clip region selected into the DC is deleted.

This function does not alter Window Manager information stored in the DC instance data. Window Manager information includes:

- The visible region
- The DC origin
- User bounds
- Cached clipping rectangles
- The HDC_IS_DIRTY flag.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
flOptions	ULONG	Option flags. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreResetDC.

flOptions Valid flags are:

RDC_SETOWNERTOSHELL When set, the graphics engine sets the ownership of the reset DC to the process that first initialized the engine, (normally the Presentation Manager interface).

RDC_RGBMODE Sets the DC's logical color table to RGB mode.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_CODEPAGE
PMERR_INV_COORDINATE
PMERR_INV_DC_TYPE
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SCAN_START

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreRestoreDC

```
#define INCL_GRE_DCS

BOOL GreRestoreDC (hdc, idDC, pInstance, lFunction)
```

This function restores the contents of a previously saved device context.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
idDC	LONG	DC state identifier. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreRestoreDC.

idDC Identifies the DC state to be restored. When this is passed as a negative value, it indicates the number of device contexts that must be popped *off* the stack to access the required DC. *−1* indicates that the most recently saved DC must be restored.

- When this parameter is passed as a positive value (*>0*) and a corresponding DC does not exist, an error is returned. The current DC is not modified.
- When passed as a negative value and there are insufficient entries on the stack, an error is raised. The current DC is not modified.
- When the value passed corresponds to a saved DC, all entries on the stack up to the one indicated are discarded. Any clip regions selected into the discarded DCs are destroyed.
- A value of *1* resets the DC stack. All entries are removed from the stack.
- A value of *0* raises an error and the DC is not modified.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_INV_COORDINATE
PMERR_INV_DC_TYPE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSaveDC

```
#define INCL_GRE_DCS

LONG GreSaveDC (hdc, pInstance, lFunction)
```

This function saves the device context's state on a stack and returns an identifier to allow for its subsequent restoration. The following information is saved:

- Current position
- Current attributes
- Current transforms, viewing limits, and clip path
- Any reference to a selected clip window
- Any loaded logical color table
- References to any loaded logical fonts
- References to the regions created on the associated DC.

The following are not saved:

- Visible region
- Process controls.

Any resources such as clip region and logical fonts, which are referenced in a saved DC, should not be deleted. The ID of a saved DC is only unique within the DC for which it is issued. Other DCs can have saved states with the same ID. The returned identifier can be used for a subsequent GreRestoreDC. This identifier represents the level of the saved DC on the save/restore stack. The first DC saved is identified by a value of *1*, the second by *2*, and so on.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	ULONG	Device context handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSaveDC

Return Codes: This function returns the identifier for the saved DC state (idDC), or GPI_ERROR if an error occurs.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_CODEPAGE
PMERR_INV_COORDINATE
PMERR_INV_DC_TYPE
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSelectBitmap

```
#define INCL_GRE_DEVSUPPORT

HBITMAP GreSelectBitmap (hdc, hbm, pInstance, lFunction)
```

This function selects a bit map into a memory DC, or (if called with a NULL bit-map handle) deselects the existing bit map from the DC. If GreSelectBitmap is called to deselect a bit map, the return code is the handle of the deselected bit map.

Once created, a bit map has to be selected into a DC before the presentation driver can write to it. A bit map can be selected into the DC that created it or into any DC that has a compatible bit-map format. Compatibility can be ensured by using one of the standard formats. Notice that the presentation driver must select a bit map into a device context before attempting to draw it.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
hbm	HBITMAP	Bit-map handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSelectBitmap

Return Codes: On completion, this function returns an HBITMAP value:

HBM_ERROR Error
Null Successful
Other Handle of deselected bit map.

An error is raised when the bit map:

- Is incompatible with the DC and cannot be converted.
- Is already selected into a DC.
- Has been assigned a Set ID for use as a pattern in an area fill operation.

Possible Errors Detected: Error codes posted by the graphics engine for this function include:

PMERR_BITMAP_IS_SELECTED
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_COORDINATE
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_HRGN
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SCAN_START

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetAttributes

```
#define INCL_GRE_DEVSUPPORT

BOOL GreSetAttributes (hdc, lPrimType, flDefsMask, flAttrsMask, pAttrs, pInstance, lFunction)
```

This function sets the attributes for the specified primitive type according to the flags set in flDefsMask and flAttrsMask. Notice that:

- Only attributes whose flags are set in flAttrsMask are modified.
- Attributes whose flags are set in both flDefsMask and flAttrsMask are set to their default values.
- Attributes whose flags are set only in flDefsMask, or whose flags are not set either in flDefsMask or flAttrsMask, are unchanged.
- Attributes whose flags are set only in flAttrsMask are set to the value specified by pAttrs.

When GreSetAttributes occurs within a path bracket, it must not be used to set the geometric line width for line attributes, nor used to set the foreground and background colors and mixes for character and marker attributes.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
lPrimType	LONG	Bundle primitive type. See below.
flDefsMask	ULONG	Flags indicating the attributes to be set to default.
flAttrsMask	ULONG	Flags indicating the attributes to be modified.
pAttrs	PBUNDLE	Pointer to the fixed-format bundle record containing the attribute values to be set. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetAttributes.

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE	Line attribute bundle
PRIM_CHAR	Character attribute bundle
PRIM_MARKER	Marker attribute bundle
PRIM_AREA	Pattern attribute bundle
PRIM_IMAGE	Image attribute bundle.

pAttrs This is a pointer to the fixed format-bundle record containing the attribute values to be set as specified by flAttrsMask. Only the attribute fields corresponding to attribute flags set in flAttrsMask, and *not* set in flDefsMask, contain valid values.

This buffer need only be large enough to contain data for the highest offset attribute referenced.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess):

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COORD_SPACE
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_END_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_UNSUPPORTED_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Remarks: When this function is called within an area definition, an error is raised by the engine and the condition PMERR_INV_IN_AREA is posted.

GreSetBitmapDimension

```
#define INCL_GRE_DEVSUPPORT

BOOL GreSetBitmapDimension (hbm, pDimension, pInstance, lFunction)
```

This function associates height and width values for the bit map indicated by hbm. These values can be read back later by calling GreGetBitmapDimension.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hbm	HBITMAP	Bit-map handle.
pDimension	PSIZEL	Pointer to width and height values in 0.1mm units. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetBitmapDimension.

pDimension Pointer to a pair of parameters:

ulWidth Width of bit map in 0.1mm units
ulHeight Height of bit map in 0.1mm units.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_INV_HBITMAP

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetBitmapID

```
#define INCL_GRE_LCID

BOOL GreSetBitmapID (hdc, hbm, lLcid, pInstance, lFunction)
```

This function sets the local identifier (lcid) for a bit map. The presentation driver needs to assign an lcid before the bit map can be used for area shading or as the pattern in a BitBlt operation. The bit map can be of any format supported by the device. However, it can be simplified by the graphics engine before use.

Errors are raised by the graphics engine when:

- The local identifier is already in use.
- The bit map is already selected into a memory DC.

Note: When a bit map is destroyed, its lcid becomes undefined.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
hbm	HBITMAP	Bit-map handle
lLcid	LONG	Local identifier to be associated with the bit map
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD=flags; low-order WORD=NGreSetBitmapID

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_INV_HBITMAP
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetDefaultArcParameters

```
#define INCL_GRE_DEVSUPPORT

BOOL GreSetDefaultArcParameters (hdc, pArcParms, pInstance, lFunction)
```

This function sets the arc parameters to the default values.

Support: This function is supported by the graphics engine and can be hooked by the presentation driver.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
pArcParms	PARCPARAMS	Pointer to the default arc parameters. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetDefaultArcParameters.

pArcParms Pointer to an ARCPARAMS structure:

lP P coefficient
lQ Q coefficient
lR R coefficient
lS S coefficient.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetDefaultAttributes

```
#define INCL_GRE_DEVSUPPORT

BOOL GreSetDefaultAttributes (hdc, lPrimType, flAttrsMask, pAttrs, pInstance, lFunction)
```

This function sets the default attributes for the specified primitive type according to the flags set in flAttrsMask. Notice that:

- Only attributes whose flags are set in flAttrsMask are modified.
- Attributes whose flags are not set in flAttrsMask are unchanged.
- Attributes whose flags are set in flAttrsMask are set to the value specified by pAttrs.

When this function occurs within a path bracket, it must not be used to set the geometric line width for line attributes. In addition, it must not be used to set the foreground and background colors and mixes for character and marker attributes.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
lPrimType	LONG	Bundle primitive type. See below.
flAttrsMask	ULONG	Flags indicating which attributes are to be modified.
pAttrs	PBUNDLE	Pointer to the fixed-format bundle record containing the attribute values to be set. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetDefaultAttributes.

lPrimType Indicates the bundle type. Valid primitive values are:

PRIM_LINE	Line attribute bundle
PRIM_CHAR	Character attribute bundle
PRIM_MARKER	Marker attribute bundle
PRIM_AREA	Pattern attribute bundle
PRIM_IMAGE	Image attribute bundle.

pAttrs This is a pointer to the fixed format-bundle record containing the attribute values to be set as specified by flAttrsMask. Only the attribute fields corresponding to attribute flags set in flAttrsMask contain valid values.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HDC_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COORD_SPACE
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_END_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_PRIMITIVE_TYPE
PMERR_INV_SETID
PMERR_UNSUPPORTED_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetDefaultViewingLimits

```
#define INCL_GRE_DEVSUPPORT

BOOL GreSetDefaultViewingLimits (hdc, prclViewingLimits, pInstance, lFunction)
```

This function sets the boundaries of the default viewing (clip) limits to the values specified by prclViewingLimits. The current viewing limits are unaffected by this call (see "GreGetViewingLimits" on page 10-67).

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
prclViewingLimits	PRECTL	Pointer to limits of viewing area. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetDefaultViewingLimits.

prclViewingLimits RECTL structure:

xLeft Minimum x-coordinate of viewing limits
yBottom Minimum y-coordinate
xRight Maximum x-coordinate of viewing limits
yTop Maximum y-coordinate.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_COORDINATE
PMERR_INV_GRAPHICS_FIELD
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetGlobalAttribute

```
#define INCL_GRE_DEVSUPPORT

BOOL GreSetGlobalAttribute (hdc, lAttrType, lAttribute, flOptions, pInstance, lFunction)
```

This function sets the specified attribute in the pen, pattern, character, image, and marker bundles. The attribute can be set to its default value or to a specified value.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
lAttrType	LONG	Specifies the attribute. See below.
lAttribute	LONG	New attribute value.
flOptions	ULONG	Option flag. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetGlobalAttribute.

lAttrType Attribute type:

ATYPE_COLOR	Foreground color
ATYPE_BACK_COLOR	Background color
ATYPE_MIX_MODE	Foreground mix
ATYPE_BACK_MIX_MODE	Background mix.

Note: ATYPE_BACK_COLOR and ATYPE_BACK_MIX_MODE do not apply to the line bundle.

flOptions The only allowable option flag is GATTR_DEFAULT, which specifies that the attribute indicated by lAttrType should be set to its default value. If the GATTR_DEFAULT flag is not set, the function sets the attribute to the value specified by lAttribute.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE	Successful
FALSE	Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_HDC_BUSY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_COLOR_ATTR
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_MIX_ATTR
PMERR_INV_RESET_OPTIONS

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetHandle

```
#define INCL_GRE_DCS

BOOL GreSetHandle (hdc, hHandle, iIndex, pInstance, lFunction)
```

This function stores a handle or variable in the device context. Up to four handles can be stored in a DC at one time. The presentation driver can only use this function for device contexts that it has created for its own use with GreOpenDC.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	ULONG	Device context handle.
hHandle	ULONG	Handle to be associated with hdc.
iIndex	ULONG	Index value of the handle in the range *0 – 3*. For a normal device context, this is a reserved parameter.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetHandle.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreSetProcessControl

```
#define INCL_GRE_DCS

BOOL GreSetProcessControl (hdc, flMask, flProcess, pInstance, lFunction)
```

This function provides a mechanism to control drawing, boundary computation and correlation. Bounds are returned in graphics model-space coordinates. If a composite transform is applied to the drawing primitives, the bounds values must be transformed back to their original values before merging with the previous bounds values.

Correlation is performed in page-coordinate space on the output of primitives that have been clipped only to the viewing limits and graphics field. Notice that correlation is performed for all functions *except* alphanumeric functions and GreErasePS. Boundary computation is performed for all functions *except* GreErasePS.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle.
flMask	ULONG	Only those flags with the corresponding bit in this parameter set are modified.
flProcess	ULONG	Process flags. See below.
pInstance	PVOID	Pointer to instance data.
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreSetProcessControl.

flProcess Process-control flags:

PCTL_DRAW	Has no effect on GreErasePS. If set, drawing primitives should appear on screen. Otherwise, output operations such as GreBitblt, GrePaintRegion, GreSetPel and other drawing primitives are not displayed.
PCTL_BOUND	Set to indicate that GPI_BOUNDS must be accumulated.
PCTL_CORRELATE	Set to indicate that correlation is to be done.
PCTL_USERBOUNDS	When set, indicates that USER_BOUNDS are to be collected for the window manager.

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_HDC_BUSY
PMERR_INV_HDC

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreStartJournalFile

```
#define INCL_GRE_JOURNALING

BOOL GreStartJournalFile (hdc, hJournal, pInstance, lFunction)
```

This function starts the journaling process. Opens the previously created journal file and turns *on* the COM_RECORDING bit. Subsequent calls to this DC drop through GreAccumulateJournalFile until GreStopJournalFile is called.

Note: The COM_DRAW bit is turned *off* until GreStopJournalFile is called.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
hJournal	ULONG	Journal file handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD=flags; low-order WORD=NGreStartJournalFile

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_BITMAP_IS_SELECTED
PMERR_COORDINATE_OVERFLOW
PMERR_DEV_FUNC_NOT_INSTALLED
PMERR_EXCEEDS_MAX_SEG_LENGTH
PMERR_HBITMAP_BUSY
PMERR_HDC_BUSY
PMERR_HUGE_FONTS_NOT_SUPPORTED
PMERR_INSUFFICIENT_MEMORY
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_BITMAP_DIMENSION
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CODEPAGE
PMERR_INV_COLOR_ATTR
PMERR_INV_COORD_SPACE
PMERR_INV_COORDINATE
PMERR_INV_DC_DATA
PMERR_INV_DC_TYPE
PMERR_INV_DRIVER_NAME
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_HJOURNAL

PMERR_INV_HRGN
PMERR_INV_ID
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_INFO_TABLE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_METAFILE
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_REF_PT_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_INV_RECT
PMERR_INV_REGION_CONTROL
PMERR_INV_SCAN_START
PMERR_JFILE_BUSY
PMERR_RAM_JNL_FILE_TOO_SMALL

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreStopJournalFile

```
#define INCL_GRE_JOURNALING

BOOL GreStopJournalFile (hdc, hJournal, pInstance, lFunction)
```

This function writes the END_OF_JOURNALFILE marker into the journal file, closes the journal file, and turns *off* the COM_RECORDING bit.

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
hdc	HDC	Device context handle
hJournal	ULONG	Journal file handle
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreStopJournalFile

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_INV_DC_DATA
PMERR_INV_HDC
PMERR_INV_HJOURNAL
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_METAFILE
PMERR_JFILE_BUSY
PMERR_RAM_JNL_FILE_TOO_SMALL

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

GreUnLoadFont

```
#define INCL_GRE_FONTS

BOOL GreUnLoadFont (pszFilename, pInstance, lFunction)
```

This function unloads the private font definitions previously loaded from the resource file indicated by pszFilename. Before unloading any fonts, the caller must ensure that:

- The fonts are not in use for the current character set.
- The relevant lcids have been deleted (see "GreDeleteSetId" on page 11-20).

Support: This function is supported by the graphics engine.

Stack Frame

Parameters	Data Type	Description
pszFilename	PSZ	Pointer to a NULL-terminated string containing the file path and name of the font file
pInstance	PVOID	Pointer to instance data
lFunction	ULONG	High-order WORD = flags; low-order WORD = NGreUnLoadFont

Return Codes: On completion, the handling routine must return BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

Possible Errors Detected: When an error is detected, the graphics engine calls WinSetErrorInfo to post the condition. Reasons for failure of this function include:

PMERR_BASE_ERROR
PMERR_FONT_FILE_NOT_LOADED

Refer to *Appendix B* of the *OS/2 2.0 Presentation Manager Programming Reference* for further explanation.

Chapter 12. System Functions

This chapter describes system functions that are available to presentation drivers:

GetDriverInfo

```
LONG APIENTRY GetDriverInfo (hEngObject, ulIndex, hdc)
```

This function returns a driver object handle for the engine object identified by hEngObject. The parameter hEngObject can be a DC handle or an engine bit-map handle (for example, the source handle, which would be passed to the presentation driver's GreBitblt routine). If hEngObject is a DC, the return code is a pointer (pInstance) to the instance data of that DC. If hEngObject is a bit map, the return code is the driver's handle for the bit map (that is the handle returned to the presentation driver when the bit map was created).

GetDriverInfo includes a check to ensure that the object is, or was created by, an instance of a DC for the same device as the DC identified by hdc. If the check is not successful, the function returns GPI_ALTERROR. GetDriverInfo is exported by the graphics engine at ordinal 30 and, if the presentation driver wants to call this function, it must be imported by the driver's module definition file.

Parameters

Parameters	Data Type	Description
hEngObject	ULONG	Handle of a DC or bit map
ulIndex	ULONG	Has the following settings: **Index = 0** hEngObject is a DC handle **Index = 1** hEngObject is a bit-map handle
hdc	ULONG	Device context handle, which identifies the calling DC

Return Codes: If successful, the function returns the instance pointer associated with the object DC or the presentation driver's handle to the object bit map. If an error is detected, the function returns GPI_ALTERROR.

SetDriverInfo

```
LONG APIENTRY SetDriverInfo (hDrvObject, hEngObject, ulIndex, hdc)
```

This function associates a driver object handle with the object identified by an engine object handle. Object handles are stored in the bit maps when they are created. Calling SetDriverInfo provides a method the presentation driver can use to change the driver handle in the bit map. SetDriverInfo includes a check to ensure that the object is, or was created by, an instance of a DC for the same device as the DC identified by hdc. If the check is not successful, the function returns GPI_ALTERROR.

SetDriverInfo is exported by the graphics engine at ordinal 31. If a presentation driver calls this function, it must be imported by the driver's module definition file.

Parameters

Parameters	Data Type	Description
hDrvObject	ULONG	Driver's handle for the object. This value must not be *–1*.
hEngObject	ULONG	Engine's handle for the object. This value must not be *–1*.
ulIndex	ULONG	Has the following settings: **lIndex = 0** Reserved. **lIndex = 1** hObject is a bit-map handle.
hdc	ULONG	Device context handle.

Return Codes: If successful, the function returns the driver object handle that was associated with the object before any changes were made by SetDriverInfo. If an error is detected, the function returns GPI_ALTERROR.

SSAllocMem

```
ULONG APIENTRY SSAllocMem (pBaseAddress, ObjectSize, Flags)
```

This function allocates a shared memory object that is managed by the selector server component of the graphics engine. Display drivers should use SSAllocMem to allocate memory for objects such as bit maps and regions. This is necessary to ensure that the returned memory object is a *global* memory object for which the system can set a new owner, or which can be marked as having no owner.

Parameters

Parameters	Data Type	Description
pBaseAddress	PVOID	Pointer to a variable, which will receive the base address of the allocated memory.
ObjectSize	ULONG	Value that specifies the size in bytes of the shared memory to allocate. The size is rounded up to the next page boundary.
Flags	ULONG	Reserved. Must be *0*.

Return Codes: SSAllocMem returns a ULONG value:

NO_ERROR
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY.

Note: Other return codes listed under DosAllocSharedMem might also apply.

SSFreeMem

```
ULONG APIENTRY SSFreeMem (BaseAddress)
```

This function frees shared memory that was allocated by a call to SSAllocMem.

Parameter

Parameters	Data Type	Description
BaseAddress	PVOID	Base address of the memory object to be freed

Return Codes: SSFreeMem returns a ULONG value:

NO_ERROR
ERROR_ACCESS_DENIED
ERROR_INVALID_PARAMETER.

VisRegionNotify

```
BOOL APIENTRY VisRegionNotify (hdc)
```

This function recalculates the visible region and validates the DC region. The handling routine in the graphics engine does the necessary calculations and calls the GreNotifyClipChange (page 8-96) routine in the presentation driver so that the driver can receive the new clip regions.

VisRegionNotify is used only by display drivers. These drivers maintain an HDC_IS_DIRTY flag that is set *on* by GreDeviceInvalidateClipRegion and cleared by GreNotifyClipChange. In the display driver, the handling routines for all drawing functions should check the HDC_IS_DIRTY flag, and if it is set, call VisRegionNotify before drawing.

VisRegionNotify is available across the system at Ring 3. Presentation drivers do not need to import this function through the module definition file.

Parameter

Parameters	Data Type	Description
hdc	ULONG	Device context handle

Return Codes: This function returns BOOLEAN (fSuccess).

TRUE Successful
FALSE Error.

WinSetErrorInfo

```
VOID cdecl WinSetErrorInfo (idError, fsOptions, arg1...argN)
```

This function posts an error message, which can be retrieved by an application that calls WinGetLastError. The syntax of the call to WinSetErrorInfo allows a variable number of parameters. The minimum requirement is the appropriate ERRORID structure and a NULL value for fsOptions.

Note: Presentation drivers do not need to import this function through the module definition file.

Parameters

Parameters	Data Type	Description
idError	ERRORID	ERRORID structure. See below.
fsOptions	ULONG	Option flags. See below.
arg1...argN	ULONG	Variable number of optional arguments. See below.

idError An ERRORID structure with the following values:

lSeverity Valid values are:

SEVERITY_ERROR
SEVERITY_NOERROR
SEVERITY_SEVERE
SEVERITY_WARNING
SEVERITY_UNRECOVERABLE

lErrorCode Error code. See "Presentation Manager Error Codes" on page 2-3 for a list of defined values.

fsOptions Option flags:

4000H Do not call DosBeep.
2000H Do not prompt the user.
0008H The next parameter is a base OS/2 error code.
0004H fsOptions must be set to 0004H to use the variable arguments arg1...argN. arg1 must contain the number of arguments that follow, not including arg1 itself. Notice that fsOptions cannot equal 000CH, that is, DOSERROR and argcount are mutually exclusive.

arg1...argN Variable number of optional arguments. These parameters have no significance except when the presentation driver is reporting a base OS/2 error. The two examples below show how WinSetErrorInfo is used to post a presentation driver error and to post a base OS/2 error:

```
Example #1.  To post a presentation driver error:

   WinSetErrorInfo (ERRORID (SEVERITY_ERROR, PMERR_INV_COORDINATE), NOBEEP);

Example #2.  To post a base OS/2 error:

   WinSetErrorInfo (ERRORID (SEVERITY_WARNING, ErrorCode), (DOSERROR+NOBEEP+NOPROMPT), DosErrorCode);
```

Return Codes: This function returns VOID.

Appendix A. Syntax Conventions

The programming statements in this book use the C language syntax. Support for code written in C is provided in header files identified by the .H file extension. Assembler support is provided in the include files identified by the .INC extension.

Parameter Names

Parameter names are constructed to show the data type of the parameter and to indicate its use:

- A lowercase prefix of one or more characters that indicates the data type.
- An optional qualifier starting with an uppercase letter.

  Where possible, standard names have been used to describe parameters. Where multiple word qualifiers are used, the order of the words is not significant.

For example:

```
hdc          /* device context handle         */
pszFilename  /* pointer to a character string */
```

The following standard base tags and their associated type names are defined:

Tag	Data Type	Description
f	BOOL	Flag or Boolean variable. The qualifier describes the condition associated with the flag when it is TRUE. For example, fSuccess is TRUE if successful, FALSE if not; whereas fError is TRUE if an error occurred, FALSE if no error occurred. For objects of type BOOL, the value 0 implies FALSE, any non-zero value implies TRUE.
ch	CHAR	Signed 8-bit quantity; a character.
s	SHORT	Signed 16-bit quantity; a SHORT. This is often used in place of us when it does not matter whether the value is signed or unsigned.
l	LONG	Signed 32-bit quantity; a LONG. This is often used in place of *ul* when it does not matter whether the value is signed or unsigned.
uch	UCHAR	Unsigned 8-bit quantity.
us	USHORT	Unsigned 16-bit quantity.
ul	ULONG	Unsigned 32-bit quantity.
b	BYTE	Unsigned 8-bit quantity; a byte. Same as *uch*.
sz	CHAR[ ]	NULL-terminated string of characters.
fb	UCHAR	Byte of flags, that is, an array of flags packed in a BYTE.
fs	USHORT	SHORT of flags, that is, an array of flags packed in a USHORT.
fl	ULONG	LONG of flags, that is, an array of flags packed in a ULONG. The three preceding types are used when more than one flag is combined into a byte, SHORT or LONG. Typically, the values are combined with the OR operator and are always unsigned.
r	REAL	Real number, single precision 32-bits.
rd	DOUBLE	Real number, double precision 64-bits.
pfn		Pointer to a function.
x		X-coordinate.
y		Y-coordinate.

The following standard prefixes are also defined:

Prefix	Description
p	32-bit pointer for an 80386 processor. For example, pch is a pointer to a character.
a	Array. For example, ach is an array of characters.
i	Index into an array. For example, an ich is used to index an ach.
c	Count. For example, cch is a count of characters.

Prefix	Description
d	Delta or difference between instances of a type. For example, dx is the difference between two values of x.
h	Handle. A value that uniquely identifies an object but cannot directly be used to access it. For example, hps is a PS handle.
mp	Mapping array. This prefix is always followed by two base types rather than just one and represents the most general case of an array. Mathematically, an array is simply a function mapping the index to the value stored in the array. mp is an abbreviation of map. In the construct mpab, *a* is the type of the index and *b* is the type of the value stored in the array. In most cases, the only type that is important is the type of the value. The index is usually an integer with no other meaning (the "a" prefix is used in this instance).
off	Offset. Generally used as an offset within a data structure. The actual address of the element within the data structure is derived by adding an offset to a pointer, which points to the beginning of the data structure. Normally, *off* is a byte offset. For example: `pfoo = (FOO *)((BYTE *)pfooBase + offfoo)`
id	Identifier. This is generally used for values that identify some object. Usually the association of the ID value and the object are established by the programmer. For example, all windows are identified by their Window ID, which can be set and queried by the programmer.
cmd	Command. Used for command values, typically as function parameters.

Some types of parameter are used in pairs; the qualifiers used reflect the relationship between the two variables. For example:

First/Last	First and last elements in a set. These are typically used with indexes or pointers (pchFirst, pchLast). Both values represent valid values (compare with Min/Max below). For all valid values of x: xFirst < = x < = xLast. The use of > with *First* or < with *Last* is almost always an *off-by-one* error.
	
	For example, to determine whether an ich is within ichFirst and ichLast:
	`if (ich >= ichFirst && ich <= ichLast)`
	`...`
	`A typical loop:`
	`for (ich = ichFirst; ich <= ichLast; ich++)`
	`...`

Min/Max	Similar to First/Last except that Max is not a valid value in the set (Min is a valid value). For all valid values of *x* in the set: xMin < = x < xMax. The use of > with Min or < = with Max is almost always an *off-by-one* error.
	
	For example, to determine whether an ich is within ichMin and ichMax:
	`if (ich >= ichMin && ich < ichMax)`
	`...`
	`A typical loop:`
	`for (ich = ichMin; ich < /* or != */ ichMax; ich++)`
	`...`
	The current value (Cur) qualifier can be used with Min and Max when Min or Max can change over time (for example, pbStackMaxCur).

Old/New	Old and new. Typically used for values or states when it is necessary to compare the old and new states of the value.
Next/Prev	Next and previous. Typically used in situations where items are being enumerated such as with linked lists.
Src/Dst	Source and destination. Typically used in transfer operations.
T	A temporary value.
Save	A temporary saved value. Typically used when saving and restoring some state.
Cur	Current value.

The base types and their prefixes are defined as follows:

Data Type	Prefix
PSZ	psz
PCH	pch
HAB	hab
HPS	hps
HDC	hdc
HRGN	hrgn
HBITMAP	hbmp
PLONG	pl
POINTL	ptl
POINTL	pt
RECTL	rcl
RECTL	rc
HWND	hwnd
WPOINT	wpt
WRECT	wrc
FIXED	fx

Parameters for defined structures are the defined parameter names. For example:

```
AREADEFS struct
         {
              defSet
              fFlags
              CodePage
         }AREADEFS
```

System-defined constants and flags are represented as two or more uppercase WORDs or mnemonic abbreviations separated by underscores. For example, SYS_CONSTANT and SYS_FLAG.

Return Values: Function handling routines pass full 32-bit return codes back to the calling function. In MASM, the return code is passed in the EAX Register.

Register Content Preservation: Registers EAX, ECX, and EDX can be destroyed. All other registers must be preserved.

Handles: All handles and pointers are 32-bit values.

Coordinates: All coordinates are passed as signed 32-bit values unless stated otherwise. World, model, and presentation-page space coordinates are restricted to the 28 low-order bits and lie within the range F8000000H through 07FFFFFFH. Device space coordinates are restricted to the 16 low-order bits and lie within the range FFFF0000H through 0000FFFFH.

Appendix B. Journal File Format

Note: *The journal file format is subject to change and should only be accessed by the engine functions. It is presented here to aid debugging of hardcopy drivers that use journal files.*

The OS/2 graphics engine journal functions create a file, either in memory or on disk. This file is used to record *all* OS/2 Presentation Manager graphics functions so that a balance can be achieved between memory usage, image detail, hardcopy device capabilities, and hardcopy device performance.

The format of the record is as follows:

Flags	Length	Function Arguments	arg data (if any)
1 WORD	1 WORD	(arg cnt DWORDs)	Variable size

LOW memory *HIGH memory*

The **arg** data is the data pointed to by any arguments in the function argument list. The actual journaled argument is changed into an offset to the journal arg data and is fixed up at playback time.

If the bit-map bits or region rects are dumped, they will be written to disk immediately following the journal record to which they belong.

The format of this additional data for bit maps is:

Data Size	Bit Map Width	Height	Planes	Bit Count	BITS
1 DWORD	1 WORD	1 WORD	1 WORD	1 WORD	*n* bytes, where n = ((bitcount*width+31)/32)*height*planes*4

The format of this additional data for regions rects is:

Data Size	DATA
1 DWORD	Data size/ 16 RECTs

The journal record for any function (including any *arg* data but not including dumped bit map or region data) is assumed to fit into the journal buffer. If regions or bit maps are written out, they are first dumped into the *region/bit-map* buffer. It is not necessary for the data to fit into this buffer all at once. However, it is assumed that regions do not have more than 64KB rectangles and bit maps do not have more than 64KB scans.

When private objects are created for the four special case functions, they are recorded. In this way, they can be destroyed by DeleteJournalFile and re-used on subsequent playbacks to the same journal file handle. If metafiling this, avoid recreating the regions from *rects* or the bit map from the bits for every play.

Appendix C. Bit Map Simulation (Hardcopy Drivers Only)

Presentation drivers for monochrome raster devices can use the system's display driver, DISPLAY.DLL, to draw the page image on a bit map. This technique reduces the amount of code in the hardcopy driver and ensures that all devices use the same drawing algorithms.

To use the display driver, the hardcopy driver has to open and manage a display DC. It must do this without invoking the graphics engine (the hardcopy driver has to act as if it were the engine). When an application opens a hardcopy DC, the Enable subfunctions in the hardcopy driver must issue appropriate calls to the Enable subfunctions in the display driver.

Note: This technique is works well for IBM display drivers. However, there is no guarantee of continued compatibility when using OEM display drivers.

These Enable subfunctions perform the following actions:

- **FillLogicalDeviceBlock**
  1. Loads the display driver and gets the address of its enable entry point.
  2. Saves a copy of the default dispatch table to pass to the display driver's FillLogicalDeviceBlock routine.
  3. Calls the display driver's enable entry point with the parameters set for FillLogicalDeviceBlock and passes a pointer to the saved default dispatch table. The display driver initializes this dispatch table and it can then be used to pass Gre*xxx* routines to the display DC.
- **FillPhysicalDeviceBlock**
  1. Saves the address of the display driver's enable entry point in the hardcopy driver's physical device block.
  2. Calls the display driver's enable entry point with the parameters set for FillPhysicalDeviceBlock and OD_MEMORY device type.
  3. Saves the value returned from the display driver's FillPhysicalDeviceBlock routine. (The value, ulStateInfo, is passed back to the display driver's EnableDeviceContext routine.)

     Note: In some source code, the name, pDCI, is used for the ulStateInfo parameter.
- **EnableDeviceContext**
  1. Calls the display driver's enable entry point with the parameters set for EnableDeviceContext. The DENPARAMS structure passed to the EnableDeviceContext routine contains the same DC handle (ulHDC) that was received by the hardcopy driver.
  2. Saves the value returned from the display driver's FillPhysicalDeviceBlock routine. This value, pInstance, needs to be passed back to the display driver on every call through the dispatch table.

     Note: In source code, pInstance might be referred to as the *magic cookie*.
- **CompleteOpenDC**
  1. Calls the display driver's enable entry point with the parameters set for CompleteOpenDC.

Any Gre*xxx* functions called from the operating system to the DC will enter the hardcopy DC through its dispatch table. Function handling routines in the hardcopy driver monitor the incoming calls and redirect those calls that affect the image through the display DC's dispatch table. When a GreEscape routine for DEVESC_NEWFRAME or DEVESC_ENDDOC is detected, the hardcopy driver transfers the bit-map bits from the display DC to a local buffer and sends them as scan lines or bands to the physical device driver.

Glossary

This glossary defines the terms used in this book. It includes terms and definitions from the *IBM Dictionary of Computing,* SC20-1699 as well as terms specific to the Presentation Manager but it is not a complete glossary for OS/2 2.0.

A

accelerator. A single keystroke that invokes an application-defined function.

action. One of a set of defined tasks a computer performs. Users request the application to perform an action in several ways: typing a command, pressing a function key or selecting the action name from an action bar or menu.

action bar. The area at the top of a panel that contains keywords that give users access to actions available in the current panel. When users select an action bar choice, a group of actions or additional keywords appear in a pull-down extension from the action bar.

action point. The current position on the screen at which the pointer is pointing. (Contrast with *hotspot* and *input focus*.)

active program. A program currently running on the computer. See also *interactive program, noninteractive program* and *foreground program*.

active window. The window with which the user is currently interacting.

alphanumeric video output. Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

anchor block. An area of Presentation Manager-internal resources allocated to a process or thread that calls WinInitialize.

anchor point. A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI. American National Standards Institute.

APA. All points addressable.

API. Application programming interface. The formally defined programming language that is between an IBM application program and the user of the program. See also GPI.

area. In computer graphics, a filled shape such as a solid rectangle.

ASCII. American National Standard Code for Information Interchange.

aspect ratio. In computer graphics, the width-to-height ratio of an area, symbol or shape.

asynchronous. (1) Without regular time relationship. (2) Unexpected or unpredictable with respect to the execution of a program's instructions.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes*.

AVIO. Advanced Video Input/Output.

B

background color. The color in which the background of a graphic primitive is drawn.

background mix. An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix*.

Bezier curves. A mathematical technique of specifying smooth continuous lines and surfaces which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bezier.

bitmap. A representation in memory of the data displayed on an APA device, usually the screen.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

button. A mechanism on a *pointing device*, such as a mouse, used to request or initiate an action. Contrast with *pushbutton* and *radio button*.

C

cancel. An action that removes the current window or menu without processing it and returns the previous window.

CASE statement. Provides, in C/2, the body of a window procedure. There is one CASE statement for each message type written to take specific actions.

cell. See *character cell*.

CGA. Color graphics adapter.

chained list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonym for *linked list*.

character. A letter, digit or other symbol.

character box. In computer graphics, the boundary that defines in world coordinates the horizontal and vertical space occupied by a single character from a character set. See also *character mode*. Contrast with *character cell*.

character cell. The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box*.

character code. The means of addressing a character in a character set, sometimes called *code point*.

character mode. The character mode in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear and angle attributes.

check box. A control window shaped like a square button on the screen, that can be in a checked or unchecked state. It is used to select one or more items from a list. Contrast with *radio button*.

check mark. The symbol (√) that is used to indicate a selected item on a pull-down.

choice. An option that can be selected. The choice can be presented as text, as a symbol (number or letter) or as an icon (a pictorial symbol).

class. See *window class*.

class style. The set of properties that apply to every window in a window class.

client area. The area in the center of a window that contains the main information of the window.

clipboard. An area of main storage that can hold data being passed from one Presentation Manager application to another. Various data formats can be stored.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits. The area of the paper that can be reached by a printer or plotter.

clipping path. A clipping boundary in world coordinate space.

code page. An assignment of graphic characters and control function meanings to all code points.

code point. Synonym for *character code*.

color dithering. A process that simulates a single color on the screen by setting alternate pels, for example, to different colors.

command. (1) The name and parameters associated with an action that can be performed by a program. A command is one form of action request. Users type in the command and enter it. (2) An action users request to interact with the command area.

command area. An area composed of two elements: a command field prompt and a command entry field.

command entry field. An entry field in which users type commands. The entry field is preceded by a command field prompt. These two elements make up the command area.

command line. On a display screen, a display line (usually at the bottom of the screen) in which only commands can be entered.

command prompt. A field prompt showing the location of the command entry field in a panel.

Common Programming Interface. A consistent set of specifications for languages, commands and calls to enable applications to be developed across all SAA environments. See also *Systems Application Architecture*.

Common User Access. A set of rules that define the way information is presented on the screen and the techniques for the user to interact with the information.

control. The means by which an operator gives input to an application. A *choice* corresponds to a control.

Control Panel. In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program. The basic function of OS/2 including DOS emulation and the support for keyboard, mouse and VIO.

control window. A class of window used to handle a specific kind of user interaction. Radio buttons and check boxes are examples.

correlation. The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

CPI. Common Programming Interface.

current position. The point from which the next primitive will be drawn.

cursor. A symbol displayed on the screen and associated with an input device. The cursor indicates

where input from the device will be placed. Types of cursors include text cursors, graphics cursors and selection cursors. Contrast with *pointer* and *input focus*.

D

data structure. (ISO) The syntactic structure of symbolic expressions and their storage allocation characteristics.

DBCS. See *double-byte character set*.

default procedure. Function provided by the Presentation Interface that may be used to process standard messages from dialogs or windows.

default value. A value used when no value is explicitly specified by the user. For example, in the graphics programming interface, the default line type is 'solid'.

Desktop Manager. In the Presentation Manager, a window from which users can start one or more listed programs.

desktop window. The window corresponding to the physical device against which all other types of windows are established.

device context. A logical description of a data destination such as memory, metafile, display, printer or plotter. See also *direct device context*, *information device context*, *memory device context*, *metafile device context*, *queued device context* and *screen device context*.

device driver. A file that contains the code needed to attach and use a device such as a display, printer or plotter.

device space. Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units.

dialog. The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box. A type of window that contains one or more controls for the formatted display and entry of data. Also known as a *pop-up window*. A modal dialog box is used to implement a pop-up window.

Dialog Box Editor. A what-you-see-is-what-you-get (WYSIWYG) editor that creates dialog boxes for communicating with the application user.

dialog item. A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog tag language. A markup language used by the DTL compiler to create dialog objects. It is based on the Standard Generalized Markup Language (SGML).

dialog template. The definition of a dialog box which contains details of its position, appearance and window ID; and the window ID of each of its child windows.

direct device context. A logical description of a data destination that is a device other than the screen (for example, a printer or plotter) and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context*.

direct manipulation. The action of using the mouse to move objects around the screen. For example, moving files and directories about in the File Manager.

directory. A type of file containing the names and controlling information for other files or other directories.

display point. Synonym for *pel*.

dithering. The process used in color displays whereby every other pel is set to one color and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work on narrow lines.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese and Korean which contain more characters than can be represented by 256 code points require double-byte character sets. As each character requires two bytes, the entering, displaying and printing of DBCS characters requires hardware and software that can support DBCS.

dragging. In computer graphics, moving an object on the display screen as if it were attached to the pointer.

drawing chain. See *segment chain*.

drop. To fix the position of an object that is being dragged by releasing the select button of the pointing device.

DTL. See *dialog tag language*.

dynamic segments. Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

E

EGA. Extended graphics adapter.

element. An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

entry field. A panel element in which users type information. Compare to *selection field.*

entry panel. A defined panel type containing one or more entry fields and protected information such as headings, prompts and explanatory text.

extended help. A facility that provides users with information about an entire application panel rather than a particular item on the panel.

entry-field control. The means by which the application receives data entered by the user in an entry field. When it has the input focus, it displays a flashing pointer at the position where the next typed character will go.

exit. The action that terminates the current function and returns the user to a higher level function. Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel.*

extended-choice selection. A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection.*

F

field-level help. Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used. The information is dependent upon the context within the work session.

File Manager. In the Presentation Manager, a program that displays directories and files and allows various actions on them.

file specification. The full identifier for a file which includes its file name, extension, path and drive.

fillet. A curve that is tangential to the end points of two adjoining lines. See also *polyfillet.*

font. A particular size and style of typeface that contains definitions of character sets, marker sets and pattern sets.

foreground program. The program with which the user is currently interacting. Also known as *interactive program.*

frame. The part of a window that can contain several different visual elements specified by the application but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles. Different standard window layouts provided by the Presentation Manager.

full-screen application. An application program that occupies the whole screen.

function key. A key that causes a specified sequence of operations to be performed when it is pressed; for example, F1 and Alt-K.

G

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics Programming Interface. The formally defined programming language that is between an IBM graphics program and the user of the program. See also *API.*

graphics. A picture defined in terms of graphic primitives and graphics attributes.

graphics attributes. Attributes that apply to graphic primitives. Examples are color, line type and shading-pattern definition. See also *segment attributes.*

graphics field. The clipping boundary that defines the visible part of the presentation-page contents.

graphics model space. The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as *model space.*

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string. See also *graphics segment.*

graphics segment. A sequence of related graphic primitives and graphics attributes. See also *graphic primitive.*

graying. The indication that a choice on a pull-down is unavailable.

group. A collection of logically-connected controls. For example, the buttons controlling paper size for a printer. See also *program group.*

H

half-toning. The conversion of colors to gray tones to simulate color on a monochrome device.

handle. An identifier that represents an object, such as a device or window, to the Presentation Interface.

hardcopy. Physical output (such as paper, slides or transparencies) from a device.

hard error. An error condition on a network that requires either that the system be reconfigured or that the source of the error be removed before the system can resume reliable operation.

heap. An area of free storage available for dynamic allocation by the program. Its size varies depending on the storage requirements of the program.

help. A function that provides information about a specific field, an application panel or information about the help facility. It provides field help when the cursor is on a selection or entry field in an application panel or another help panel. It provides information about the application panel, called *extended help*, when the cursor is not in an interactive field.

help index. A facility that allows the user to select topics for which help is available.

help panel. A panel with information to assist users that is displayed in response to a help request from the user.

help window. A Common User Access defined secondary window that displays information when the user requests help.

hit testing. The means of identifying which window is associated with which input device event.

hook. A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

hook chain. A sequence of hook procedures that are "chained" together so that each event is passed in turn to each procedure in the chain.

hotspot. The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point*.

I

icon. A pictorial representation of an item the user can select. Icons can represent items (such as a document file) that the user wants to work on and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions and minimized programs.

image font. A set of symbols each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of the symbol. Contrast with *outline font*.

information device context. A logical description of a data destination other than the screen (for example, a printer or plotter) but where no output will occur. Its purpose is to satisfy queries. See also *device context*.

information panel. A defined panel type characterized by a body containing only protected information.

input focus. The area of the screen that will receive input from an input device (typically the keyboard).

input router. OS/2 internal process that removes messages from the system queue.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

interactive program. A program that is running (active) and is ready to receive (or is receiving) input from the user. Compare with *active program* and contrast with *noninteractive program*.

Also known as a *foreground program*.

interchange file. Data that can be sent from one Presentation Interface application to another.

J

journal. A special-purpose file that is used to record changes made in the system.

K

kerning. The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keys help. A facility that gives users a listing of all the key assignments for the current application.

L

label. In a graphics segment, an identifier of one or more elements that is used when editing the segment.

LIFO stack. A data stack from which data is retrieved in last-in, first-out order.

linked list. Synonym for *chained list*.

LVB. Logical Video Buffer.

M

main window. The window that is positioned relative to the desktop window.

marker box. In computer graphics, the boundary that defines in world coordinates the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol. A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

maximize. A window-sizing action that makes the window the largest size possible.

media window. The part of the physical device (display, printer or plotter) on which a picture is presented.

memory device context. A logical description of a data destination that is a memory bitmap. See also *device context*.

menu. A type of panel that consists of one or more selection fields. Also called a *menu panel*.

message. 1. In Presentation Manager, a packet of data used for communication between the Presentation Interface and windowed applications.

2. In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process.

Messages on status, problems or user actions from a computer application should be distinguished from a "message" or note sent to users by other users over a communications link.

message filter. The means of selecting which messages from a specific window will be handled by the application.

message queue. A sequenced collection of messages to be read by the application.

metafile. The generic name for the definition of the contents of a picture. Metafiles are used to allow pictures to be used by other applications.

metafile device context. A logical description of a data destination that is a metafile which is used for graphics interchange. See also *device context*.

metalanguage. A language used to specify another language. In this publication, the data types are described using a metalanguage so as to make the descriptions independent of any one computer language.

micro presentation space. A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize. A window-sizing action that makes the window the smallest size possible. In the Presentation Manager, minimized windows are represented by icons.

mix. An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix*. Contrast with *background mix*.

mixed character string. A string containing a mixture of one-byte and *kanji* or *hangeul* (two-byte) characters.

modal dialog box. The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also known as a *serial dialog box*. Contrast with *parallel dialog box*.

modeless dialog box. The type of control that allows the operator to perform input operations on any of the application's windows. Also known as a *parallel dialog box*. Contrast with *modal dialog box*.

model space. See *graphics model space*.

mouse. A hand-held device that is moved around to position the pointer on the screen.

multitasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

N

named pipe. A named object that provides client-to-server, server-to-client or duplex communication between unrelated processes. Contrast with *unnamed pipe*.

noninteractive program. A program that is running (active) but is not ready to receive input from the user. Compare with *active program* and contrast with *interactive program*.

nonretained graphics. Graphic primitives that are not remembered by the Presentation Interface once they have been drawn. Contrast with *retained graphics*.

null-terminated string. A string of (n + 1) characters where the (n + 1)th character is the 'null' character (X'00') and is used to represent an n-character string with implicit length. Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object window. A window that does not have a parent but which may have child windows. An object window cannot be presented on a device.

open. To start working with a file, directory or other object.

outline font. A set of symbols each of which is created as a series of lines and curves. Contrast with *image font*.

output area. The area of the output device within which the picture is to be displayed, printed or plotted.

owner window. A window into which specific events that occur in another (owned) window are reported.

P

page viewport. A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation page contents are transformed automatically to the page viewport in device space.

paint. The action of drawing or redrawing the contents of a window.

panel. A particular arrangement of information grouped together for presentation to the user in a window.

panel area. An area within a panel that contains related information. The three major panel areas defined by the Common User Access are the action bar, the function key area and the panel body.

panel body. The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body may contain protected information, selection fields and entry fields. The layout and content of the panel body determine the panel type.

panel body area. The part of a window not occupied by the action bar or function key area. The panel body area may contain information, selection fields and entry fields. Also known as *client area*.

panel body area separator. A line or color boundary that provides users with a visual distinction between two adjacent areas of a panel.

panel definition. A description of the contents and characteristics of a panel. Thus, a panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID. A panel element located in the upper left-hand corner of a panel body that identifies that particular panel within the application.

paper size. The size of paper, defined in either standard U.S. or European names (for example, A, B, A4) and measured in inches or millimeters respectively.

parallel dialog box. See *modeless dialog box*.

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonymous with *display point*, *pixel* and *picture element*.

pick. To select part of a displayed object using the pointer.

picture chain. See *segment chain*.

picture element. Synonym for *pel*.

pipe. See *named pipe*, *unnamed pipe*.

pixel. Synonym for *pel*.

plotter. An output device that uses pens to draw its output on paper or transparency foils.

pointer. The symbol displayed on the screen that is moved by a pointing device such as a *mouse*. The pointer is used to point at items that users can select. Contrast with *cursor*.

pointing device. A device (such as a mouse) used to move a pointer on the screen.

pointings. Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device such as a *mouse*.

polyfillet. A curve based on a sequence of lines. It is tangential to the end points of the first and last lines and tangential also to the midpoints of all other lines. See also *fillet*.

polyline. A sequence of adjoining lines.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up window. A window that appears on top of another window in a dialog. Each pop-up window must

be completed before returning to the underlying window.

Presentation Manager. The OS/2 Control Program plus the visual component that presents, in windows, a graphics-based interface to applications and files installed and running in OS/2.

presentation page. The coordinate space in which a picture is assembled for display.

presentation space (PS). Contains the device-independent definition of a picture.

primary window. The window in which the main dialog between users and the application takes place. In a multi-programming environment, each application starts in its own primary window. The primary window remains for the duration of the application although the panel displayed will change as the user's dialog moves forward. See also *secondary window*.

primitive. See *graphic primitive*.

primitive attribute. A specifiable characteristic of a graphic primitive. See *graphics attributes*.

print job. The result of sending a document or picture to be printed.

Print Object. In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

process. An instance of an executing application and the resources it is using.

program group. In the Presentation Manager, several programs that can be acted upon as a single entity.

program name. The full file specification of a program. Contrast with *program title*.

program title. The name of a program as it is listed in the Desktop Manager window. Contrast with *program name*.

push. To add an item to a last-in-first-out stack of items. Contrast with *pop*.

pushbutton. A control window shaped like a rounded-corner rectangle and containing text, that invokes an immediate action such as 'enter' or 'cancel'.

Q

queue. A list of print jobs waiting to be printed.

queued device context. A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context*.

R

radio button. A control window shaped like a round button on the screen that can be in a checked or unchecked state. It is used to select a single item from list. Contrast with *check box*.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase. A word or phrase that is emphasized in a device-dependent manner in order to inform the user that additional information for the word or phrase is available.

reference phrase help. Provides help information on a selectable phrase.

refresh. To update a window with changed information to its current status.

region. A clipping boundary in device space.

resource. The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators and mnemonics; the definitions are held in a resource file.

restore. To return a window to its original size or position following a sizing or moving action.

retained graphics. Graphic primitives that are remembered by the Presentation Interface after they have been drawn. Contrast with *nonretained graphics*.

reverse video. A form of alphanumeric highlighting for a character, field or cursor in which its color is exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB. Red-green-blue. For example "RGB display".

S

screen. The physical surface of a workstation or terminal upon which information is presented to users.

screen device context. A logical description of a data destination that is a particular window on the screen. See also *device context*.

scrolling. Moving a display image vertically or horizontally in a manner such that new data appears at one edge as existing data disappears at the opposite edge.

secondary window. A type of window associated with the primary window in a dialog. A secondary window

begins a secondary and parallel dialog that runs at the same time as the primary dialog.

segment. See *graphics segment.*

segment attributes. Attributes that apply to the segment as an entity as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain. All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonymous with *picture chain.*

segment store. An area in a normal graphics presentation space where retained graphics segments are stored.

select. To mark or choose an item. Notice that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

semaphore. An object used by multi-threaded applications for signaling purposes and for controlling access to serially reusable resources.

separator. See *panel body area separator.*

serial dialog box. See *modal dialog box.*

serially reusable resource (SRR). A logical resource or object that can be accessed by only one task at a time.

session. A routing mechanism for user interaction via the console.

shear. The tilt of graphics text when each character leans to the left or right while retaining a horizontal baseline.

shortline. A collection of monotonically increasing x-values representing the first of each group of x-values associated with each y-value.

shutdown. In the Desktop Manager, the procedure required before the computer is switched off to ensure that data is not lost.

spline. A sequence of one or more Bezier curves.

spooler. A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete and the required device is available. The spooler prevents output from different sources being intermixed.

standard window. A collection of windows that form a panel.

style. See *window style.*

suballocation. The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

symbolic identifier. A text string that equates to an integer value in an include file that is used to identify a programming object.

system queue. This is the master queue for all pointer device or keyboard events.

Systems Application Architecture. A formal set of rules that enables applications to be run without modification in different computer environments.

T

template. An ASCII-text definition of an action bar and pull-down menu held in a resource file or as a data structure in program memory.

text. Characters or symbols.

text window. Also known as the VIO window. The environment in which OS/2 runs AVIO applications.

thread. A unit of execution within a process.

transform. (1) The action of modifying a picture by scaling, shearing, reflecting, rotating or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation.*

U

unnamed pipe. A circular buffer created in memory; used by related processes to communicate with one another. Contrast with *named pipe.*

update region. A system provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window that are visually invalid or incorrect and therefore in need of repainting.

User Shell. A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

V

vector font. A set of symbols, each of which is created as a series of lines and curves. Contrast with *image font* and *outline font.*

VGA. Video graphics array.

viewing pipeline. The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window. Clipping boundary that defines the visible part of model space.

VIO. Video Input/Output.

virtual memory (VM). Addressable space that is apparent to the user as the processor storage space but not having a fixed physical location.

visible region. A window's presentation space clipped to the boundary of the window and the boundaries of any overlying window.

W

wild-card character. The global file-name characters ? or *.

window. A rectangular area of the screen through which a panel or portion of a panel is displayed. A window can be smaller than or equal in size to the screen. Windows can overlap on the screen and give the appearance of one window being on top of another.

window class. The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates. The means by which a window position or size is defined; measured in device units or *pels*.

window rectangle. The means by which the size and position of a window is described in relation to the desktop window.

world coordinates. Application-convenient coordinates used for drawing graphics.

world-coordinate space. Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG. What You See Is What You Get. A capability that enables text to be displayed on a screen in the same way that it will be formatted on a printer.

Z

zooming. In graphics applications, the process of increasing or decreasing the size of picture.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

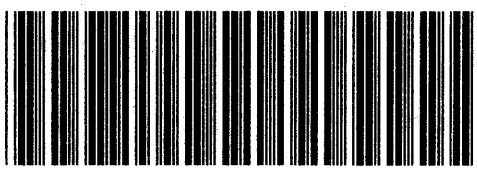
W

Numerics

International Business
Machines Corporation

Printed in the
United States of America

10G6267



S10G-6267-00

P10G6267