Microsoft®
Operating System/2

Presentation Manager Reference

Addendum

Microsoft Corporation

Document No. 060060011-103-R00-0388
Part No. 02900

Contents

Figures

Preface

The *Microsoft Operating System/2 Presentation Manager Reference Addendum* is derived from a draft of the functional specification of the MS OS/2 Presentation Manager and contains selected chapters from the *Microsoft Operating System/2 Presentation Manager Reference* described in the *Microsoft Operating System/2 Presentation Manager Startup Guide.* This addendum describes the Presentation Manager shell, Icon Editor, Font Editor, and Dialog Editor—information that is not present in the *Microsoft Operating System/2 Programming Reference Manual.* To help you find this information, chapter numbers were preserved to match those described in the *Microsoft Operating System/2 Presentation Manager Startup Guide.* The *Microsoft Operating System/2 Programming Reference Manual* is a more up-to-date and accurate draft of the MS OS/2 1.1 application programming interface and replaces Chapters 4 through 14 of the *Microsoft Operating System/2 Presentation Manager Reference.*

Although this addendum does not represent the final Presentation Manager specification, it does provide a reasonable preview of the functionality you can expect from the final product. This documentation is preliminary in nature. The application program interface and other features of the Presentation Manager described in this document are subject to change.

Chapter 1
Introduction and Guide to the Presentation Manager

1.1 Introduction and Guide to Presentation Manager

This section introduces Presentation Manager to the end user.

1.1.1 What Is Presentation Manager?

Presentation Manager is an extension of the basic MS OS/2 operating system. Presentation Manager is the 'Presentation Manager' component of MS OS/2. Its features include:

- the ability to view output from multiple applications on the display simultaneously
- an enhanced User Interface to both MS OS/2 and application functions
- programming interfaces which provide applications with sophisticated functions:
  - for generating and displaying Graphics and Alphanumerics data on a range of output devices including the display screen.
  - for handling Input devices such as mouse and keyboard
  - for Windowing data onto the display screen
  - for the provision of a User Interface which is both rich in function and consistent across applications.

Applications which run with the basic MS OS/2 will also run when Presentation Manager is present. However, not all these applications can take advantage of the additional facilities provided by Presentation Manager. In particular, applications which attempt to access the display or input devices directly cannot share the screen concurrently with other applications and cannot use the Presentation Manager programming interfaces. The applications which cannot take advantage of Presentation Manager are termed non-Presentation Manager Applications.

1.1.2 Fundamental Features of Presentation Manager

1.1.2.1 User Interface Shell

When the MS OS/2 system is started up with Presentation Manager present, the display screen is initially occupied by the Presentation Manager User Interface Shell. The Presentation Manager User Interface Shell replaces the simple User Interface Shell provided with MS OS/2. It provides the following end-user functions:

Start an Application

The user is presented with a list of all the available applications and can choose one to start. There is a 'command line' option, which enables the user to start a program by entering the command line in a manner consistent with MS OS/2.

A means is provided for the user to update the list of of applications - adding or removing entries as desired - and updating the application profile for each of them.

Switch to another Application

The user can display all the applications which are running and can select which one to work with next. The list includes both Presentation Manager and non-Presentation Manager applications.

Control of the Position and Size of Application Windows

Each Presentation Manager application has one or more Windows on the screen. The User Interface Shell provides the user means of controlling the size and position of the windows visible on the screen.

Control of the Printing functions

A menu is provided to give the user control over the Printing functions performed by Presentation Manager.

Use of MS OS/2 file system

An easy-to-use method of interacting with the MS OS/2 file system is provided, that allows the end-user to perform file commands such as copying or renaming files.

Control functions

Provides the user with a consistent and easy-to-use method of selecting defaults for various Presentation Manager parameters, e.g., the color of empty space on the screen.

1.1.2.2 Screen Appearance

As applications are started by the user they appear on the screen. The applications fall into two classes - Presentation Manager and non-Presentation Manager. For Presentation Manager applications, the User Interface Shell menus remain visible until explicitly removed by the user. For non-Presentation Manager applications, the User Interface Shell disappears when the application is on the screen.

Non-Presentation Manager applications are not able to take advantage of the features of Presentation Manager.

Presentation Manager applications are able to take full advantage of the features of the Presentation Manager functions. These applications do not have to use the Presentation Manager unique programming interfaces but do have to obey rules concerning their use of the display screen and the input devices. Put simply, when using the display and input devices an application must use the Presentation Manager programming interfaces and/or use the basic MS OS/2 VIO.., KBD.. or MOU.. function calls.

When the user wants to interact with a non-Presentation Manager application, the application always appears on the screen by itself. Non-Presentation Manager applications cannot share the screen with other applications. Neither can they share the screen with the User Interface Shell. Thus the application cannot be seen at all when the user is interacting with another application or the User Interface Shell.

The User Interface Shell and all the Presentation Manager applications occupy the Presentation Manager Screen Group. They can all potentially appear on the screen simultaneously, in overlapped windows. A Window is a rectangular region on the screen within which application data is displayed. The Presentation Manager screen has a 'Messy Desk' appearance in that the rectangular windows can overlap one another. Where the windows overlap, only part of one window is displayed and the appearance is like that of papers on a desktop, i.e., one piece of paper overlays another and only the topmost one can be seen where they overlap.

A simple example of the Presentation Manager Screen Group appearance is shown in the following diagram.

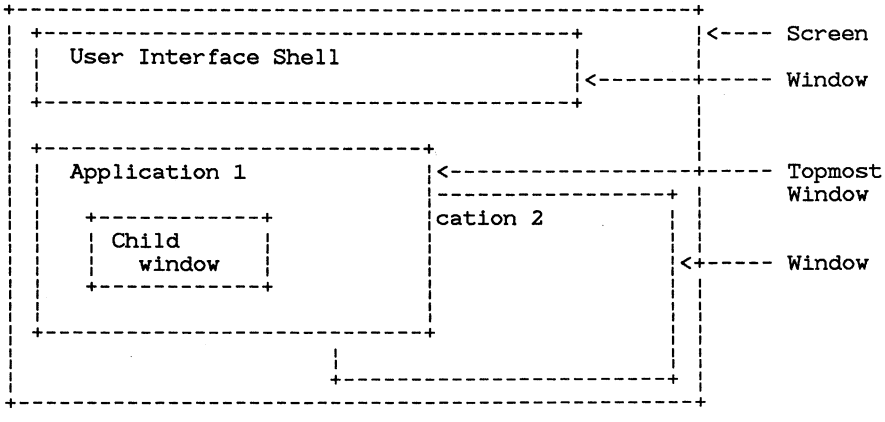


Figure 1.1 Typical Presentation Manager Screen Layout

A Presentation Manager application generally has one window and can have many more. Windows are organized on a hierarchical parent-child basis. A child window always lies on top of and is contained within its parent window. The windows at the top of the structure (which can be thought of as children of the physical screen) are called *top-level* windows. An application may have one or more top-level windows.

The top-level window with which the user is interacting is called the *active* window. This will lie visually on top of all other top-level windows. Keyboard input is always directed to the input focus window. The input focus window is either the active window or a child of the active window.

The mouse input is generally directed to the window that lies underneath the mouse pointer.

Some user input is received by the User Interface Shell rather than an application. This input generally performs operations beyond the scope of a single application, such as allowing the user to switch the active window. Certain keys on the keyboard and the mouse cause this kind of input. A detailed description is provided in the section dealing with the User Interface Shell.

1.1.2.3 The Pointer

Part of the screen appearance related to input is the Pointer. The pointer is a small image which moves around the screen as the mouse is moved. It is displayed only on those systems which have a mouse attached. It appears on top of anything else displayed on the screen.

Its appearance can vary. There is a System Pointer appearance, an arrow, which the pointer has by default. The shape can change when the pointer enters an application window. The pointer shape can also vary as it moves from selectable to non-selectable items on the screen.

The position of the Pointer on the screen is termed the Action Point. The Pointer can be used to select objects by positioning the pointer over the object and pressing and releasing one of the mouse buttons. Since the pointer is generally a large object, the action point occupies a point within the pointer shape. This point must be chosen carefully to avoid confusing the user. For instance, the action point of the System Pointer is at the tip of the arrow.

1.1.2.4 Presentation Manager Windows

Presentation Manager windows are more than just simple rectangles on the screen. They have a number of optional features which occupy their borders, termed the *frame window.* The frame window gives the end-user access to a number of Presentation Manager functions. The frame window includes:

- Borders
- Title Bar
- Scroll Bar
- Menu Bar
- System icon
- Maximize and minimize icons

The area in the center of the window that would normally contain the main information content of the window is called the *Client area.*

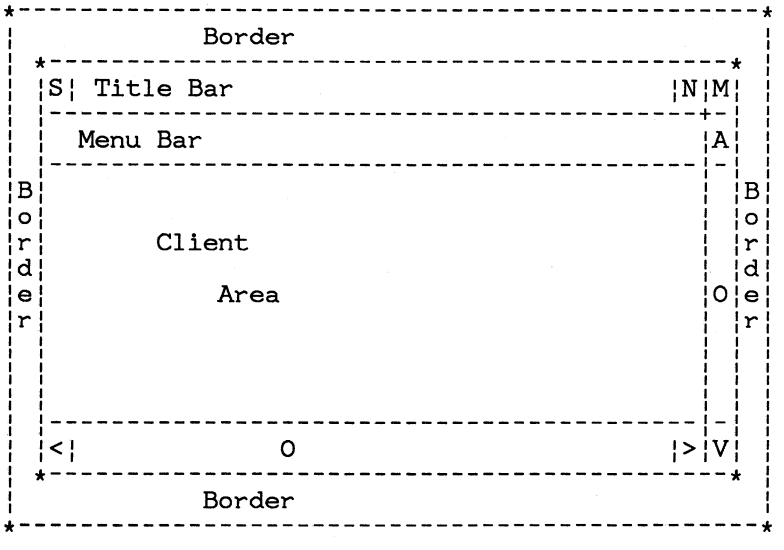


Figure 1.2 Presentation Manager Window with Frame

Window Border

Presentation Manager windows have a border in one of four formats:

- Normal border (that is not selectable by the user)
- Heavy border (that is selectable by the user for operations such as sizing a window)
- A thin border (that is not selectable)
- No border

Title Bar

The title bar is the window name that appears at the top of a window. Highlighting of the title bar indicates the window with which the user is currently interacting.

Scroll Bars

A window can contain one or two optional Scroll Bars. There is a Vertical Scroll Bar which appears at the right of the window and a Horizontal Scroll Bar which appears at the bottom of the window. The scroll bars can be used to move the data appearing in the window up and down or right and left, under either application or Presentation Manager control.

Menu Bar

A menu bar is a horizontally aligned menu at the top of the window. The end user may make selections on the menu bar that either send commands directly to the application, or cause the selection of a pull-down menu.

System icon

The system icon is an icon that the user may select in order to activate the system menu for the window. The system menu contains functions such as move, size.

Maximize icon

The maximize icon is an icon that the user may select in order to change a window to its maximum size.

Minimize icon

The minimize icon is an icon that the user may select in order to change a window to its minimum size.

1.1.2.5 Presentation Manager User Controls

The Presentation Manager User Controls provide the application program with consistent means of interacting with the user to perform various standard operations. These are:

- Interaction by use of menus
- Interaction by use of dialog boxes

1.1.2.5.1 Use of Menus

The use of menus to interact with an application will always commence with a menu bar. The menu bar is a horizontal bar along the top of a window that contains a number of items. The items may be selected, one at a time, by the user. The selection of an item in the menu bar by the user will cause the appearance of a secondary menu, called a pull-down menu. The pull-down menu contains additional options, one or more of which may be selected by the user. On completion of the selection, the pull-down menu is removed and the application performs the required action.

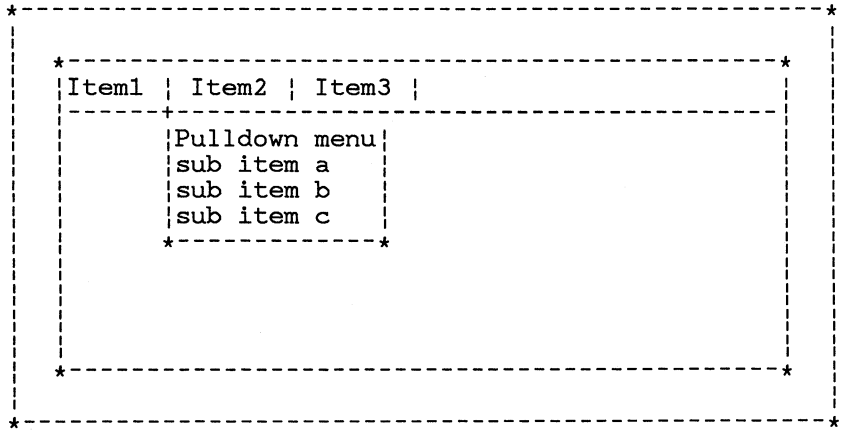


Figure 1.3 Menu Bar with Pull-down Menu

1.1.2.6 Presentation Manager Programming Functions

Presentation Manager has a large Application Programming Interface which is subdivided into major functional groups:

- Windowing - creation and control of windows within an application
- Input and Message Handling
- User Controls
- Alphanumerics Output
- Graphics Output
- Bitmaps
- A programmed interface to the User Interface Shell

It is not necessary for an application to use any of the Presentation Manager API functions in order to run as part of the Presentation Manager Screen Group and be windowed onto the screen with other applications. An application using the VIO.., KBD.. and MOU.. functions of basic MS OS/2 can be windowed when Presentation Manager is present. No changes to the application are necessary.

However, an application using the Presentation Manager API has access to a range of powerful functions which can enhance the functionality and usability of the application while at the same time reducing the effort required to produce it.

A summary of the groups of functions in the Presentation Manager API follows.

Windows

> An application can create and use a number of windows on the screen via the Windows API. Function is provided to control the size and position of a window and also to control whether the user can size or position a window. The application can specify the form of the window frame. The application can also control the data which appear in each window and can control which window is the Input Window.

User Interface Controls

> The User Interface Controls API provides the application with functions for dialog between application and the user. The functions include:
>
> - The display and interaction with menus.
>
> The following menus are supported:

- Menu Bars
- Pull-down menus

- Control functions that an application would typically group together into a 'dialog box'. These are:
  - Scroll bars
  - Buttons
  - Edit controls
  - Static controls
  - List boxes
  - Message boxes

Input and Message Handling

The Input API allows the application to control the input it receives, both from the user via the Mouse and Keyboard and from the system and other applications in the form of messages. The input is based on an application input queue, and one or more Window processing functions.

Alphanumerics Output

The Alphanumerics output API, termed Advanced Vio, is used to output simple Alphanumeric data into screen Windows or into Bitmaps. Advanced Vio is an extension of the basic MS OS/2 VIO.. functions for a windowing environment. Advanced Vio also allows use of multiple fonts and features such as underscoring of individual characters.

Graphics Output

The Graphics API, called the GPI, is used to draw graphics data into screen windows, bitmaps, or other devices such as printers and plotters. The application can draw a range of graphics objects, such as Lines, Arcs, Text Strings, Closed Areas and Images. Various attributes of the primitives such as their Color, Area Fill pattern, Character Font and Line Style can be controlled. The size, orientation and position of every primitive can be varied by means of Transformations.

The GPI also supports a wide range of text functions, including the support of fonts in a manner consistent with the Font Object Content Architecture.

Graphics data may be managed by the application or retained and managed by the Presentation Manager system.

Bitmaps

The Bitmap API allows creation and use of Bitmaps. Bitmaps are best thought of as images similar in form to the screen

image. Bitmaps can be drawn into in a similar fashion to the screen; they may reside either in PC memory or in memory associated with a particular device. Bitmaps can also be the source of data to place on the screen. They can be used to produce rapid changes to the screen, such as changing a Menu, in cases where normal drawing would be too slow.

User Interface Shell API

Presentation Manager contains an API that will allow applications to request some of the shell functions normally requested by the user.

Chapter 2
Application Model

2.1 How to Write a Presentation Manager Application—A Guide to the API

This section describes how to write a Presentation Manager application. It describes the environment in which a Presentation Manager application runs and gives a guide to the concepts and methods of using the Presentation Manager API. A detailed description of the Presentation Manager API is in the later sections.

2.1.1 The Purpose of Presentation Manager and Its API

The basic purpose of Presentation Manager is to provide easy accessibility for the user to the functions provided by the PC system. It does this in conjunction with MS OS/2 and together they help the user accomplish whatever tasks need doing, as and when they are needed.

An important feature of MS OS/2 and Presentation Manager is that of multi-tasking. The system can perform a number of tasks simultaneously - multiple application programs can run at the same time. Presentation Manager allows the user to see the data belonging to many applications simultaneously, so that one set of data can be used in conjunction with another.

Presentation Manager makes it easy to get things done. In contrast to the situation on MS OS/2, there is no need to stop one application program just because the user needs to run another to get access to some piece of information. Presentation Manager provides means to start any task at any time. It provides means to view many tasks simultaneously on the screen. Presentation Manager also provides means for copying data from one task to another ("Cut and Paste") which really makes the use of multiple tasks interesting and useful.

For application programs, Presentation Manager provides *shared* access to the general resources of the PC system, which include:

- The Screen
- The Keyboard
- The Mouse
- Printer(s)
- Plotter(s)
- Picture Files
- Other Applications

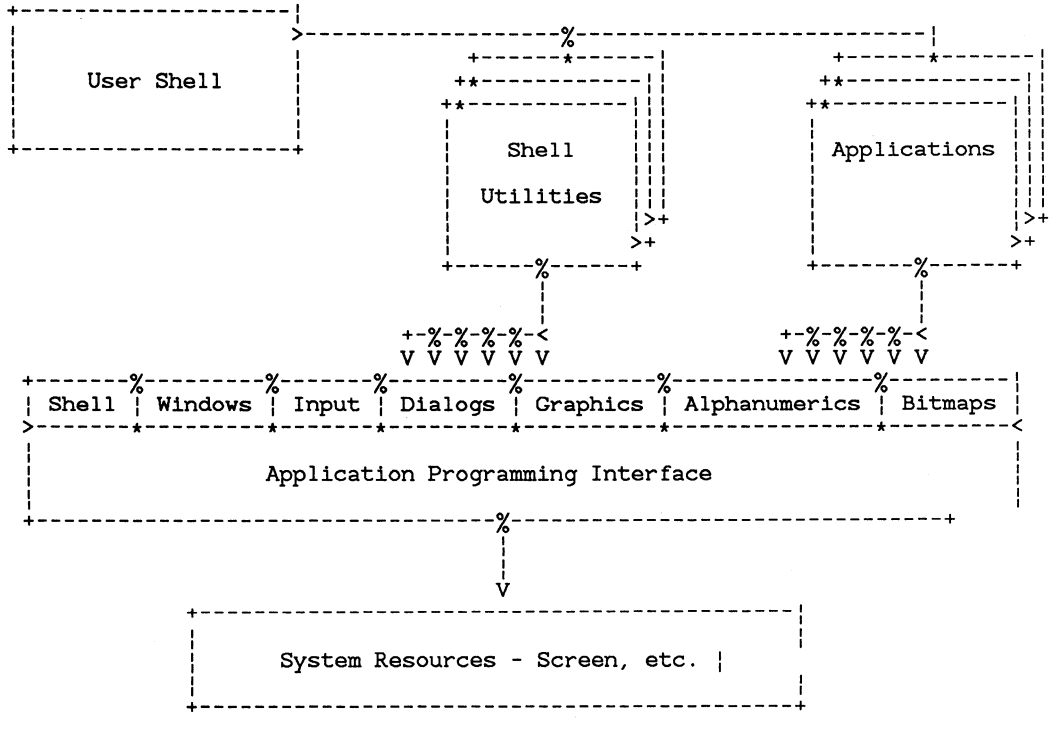
Presentation Manager makes the access to the resources simple. Presentation Manager also simplifies the process of writing an application through the functions provided in the API and the various utility programs provided in the Toolkit.

The Presentation Manager API does require applications to behave in certain ways, in order to share resources effectively. In simple terms, applications wishing to take advantage of the Presentation Manager features must behave in a *cooperative* fashion. Applications must cooperate with both the Presentation Manager system and other applications in order that the system works to the benefit of the end user.

The Presentation Manager API is structured around these basic ideas and does make some demands on the way applications work. This is made clear in later sections.

MS OS/2 applications that do not wish to take advantage of the Presentation Manager facilities can run in a PC system that is using Presentation Manager. How such applications run in a system using Presentation Manager is dealt with in a later section.

2.1.1.1 Presentation Manager Basic System Structure



2.1.1.1.1 The User Interface Shell

In the user's eyes, the major part of the Presentation Manager system is the User Interface Shell. The User Interface Shell presents the user with a view of the various components of the system. It allows the user to:

- get tasks (applications) running
- work with a particular application
- control the layout of applications on the screen, including position and visibility
- view and work with the data files in the system
- control the Printer(s) and Plotter(s) attached to the system
- control various aspects of the appearance and operation of the system, such as the screen colors

The User Interface Shell is designed to make the system easy to understand and easy to use. It gives rapid access to the capabilities of the system. At the same time, it is functionally very rich and caters for the expert user. Applications should be written with the functions of the User Interface Shell in mind - to avoid unnecessary duplication.

It is important that the various parts of the User Interface Shell and the application programs in the system have the same Style. This means that they are uniform in appearance and all work in the same kind of way, even though the function provided by different applications may be very different. This means that the user does not have to jump from one environment to another and can proceed with ease from one task to the next. In fact, the user should not really be aware of moving from one application to another.

The Presentation Manager API makes it easy for a program to conform to a standard Style. This is discussed in detail in a later section.

Thus, the User Interface Shell provides access to the system, to various utilities and to the applications installed in the system.

2.1.1.1.2 The API

Applications access the functions of the Presentation Manager system via the API. The API and its associated Utilities simplify the process of writing an application. It provides the following broad areas of function:

- Display of *Data* on the *Screen* and on *Printers* and *Plotters*. The data may be simple Text ('Alphanumerics') or Graphics (including high quality Text).
- Presentation and Operation of Standard User Menus and Dialogs on the screen to aid the user in accomplishment of some task.
- Interaction with other Functions or Applications in the system, including Shell processes and functions.
- User Interaction and Input functions.
- Partitioning of Screen data, economical use of Screen area and structuring of application functions.

The description of the API is divided into a number of functional areas:

Shell access to aspects of the User Interface and to the Utilities that form part of the User Interface Shell, including:

- Starting of Programs - Program Names
- Listing of running applications
- Clipboard - copying of data between programs
- Program environment information - initial values for position and size of an application, for example

Windows involves provision of areas on the screen in which to draw data. However, the function is much more extensive than this, and touches on:

- program structuring including object-oriented programming
- user interface functions
- user input
- inter-program communication.

Input which covers:

- user input from Mouse and Keyboard
- system messages and inter-application messages
- timer functions

Dialogs and Menus
which includes:

- Display and operation of Menus offering the user straightforward selections from a list of items.
- Creation, display and operation of Dialogs which offer the user more complex forms of interaction with the application.

Alphanumerics Output
which is the output of simple textual information to the screen, printers, plotters and files. This offers a way of displaying text in a simple form as fast as possible.

Graphics Output
which allows the application to create and display graphical data on the screen, printers, plotters and files. This includes high quality and high function typographical text functions.

Bitmaps
which allow the creation of bitmapped graphical images for the purpose of rapid manipulation of the appearance of the screen.

2.1.2 API General Features

The Presentation Manager API provides functions for the Interface between an application program and the user sitting in front of a PC. For most applications, this means:

- *Output Functions* for the presentation of data of various forms on the Screen in a consistent manner.
- *Input Functions* for the handling of user requests and responses.

Devices other than the screen are also supported for data output, such as Printers, Plotters and Data Files. However, these devices do not participate directly in the interface between the application and the user.

Presentation Manager organizes its user related functions, both output and input, around *Screen Windows.* An application can create as many windows as it desires. Each window serves a part in the dialog between application and user. One window at a time is the center of attention for the user, although other windows may be visible and convey useful and important information.

The way in which an application uses the Presentation Manager API is summarized in:

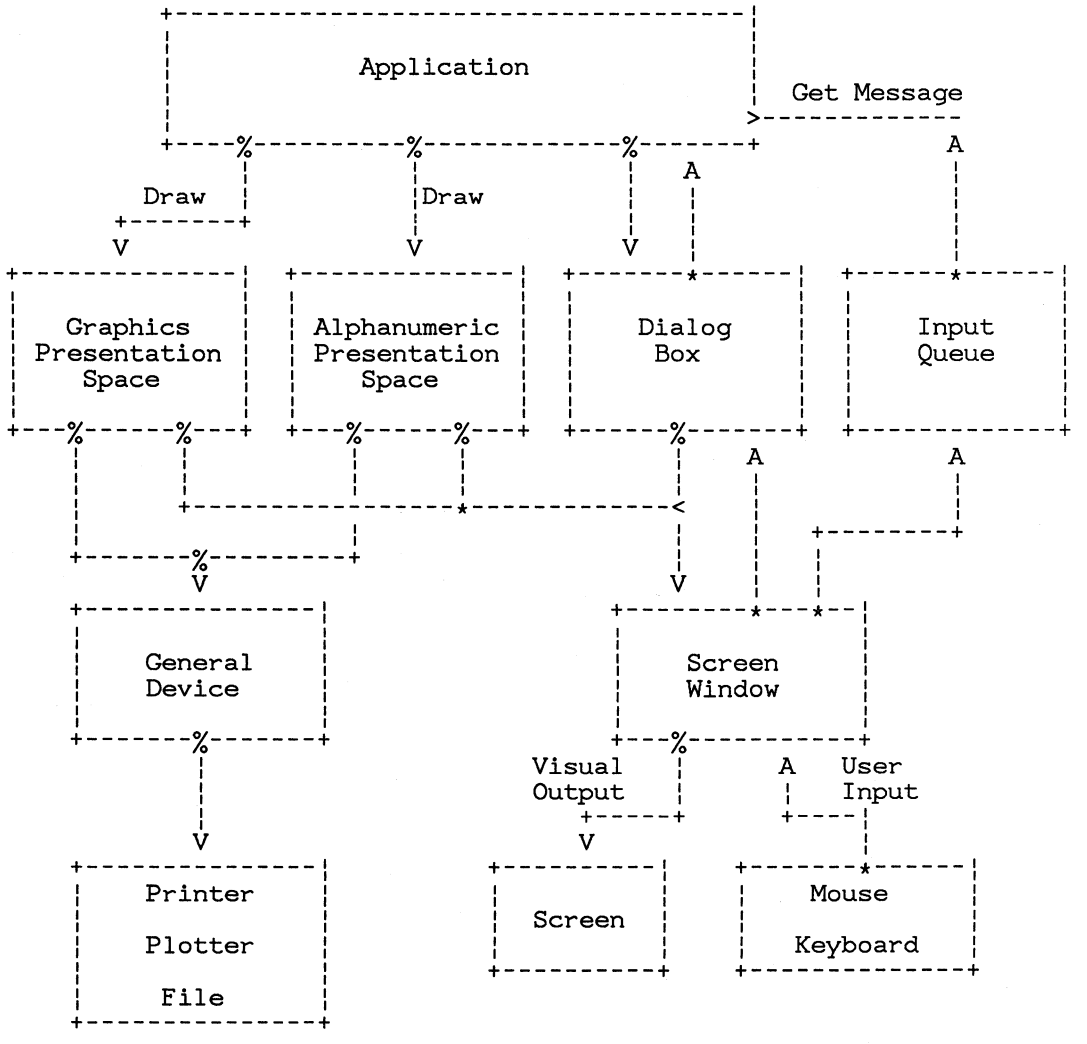


Figure 2.1 API Functions—Application Model for Input and Output

2.1.2.1 Output Fundamentals

2.1.2.1.1 Output Data

The application creates output data in one of three forms:

Alphanumeric Data
: which is Text and Numeric data displayed on a fixed grid of character 'slots'. These are held in an Alphanumeric Presentation Space.

Graphics Data
: such as lines, circles and shapes filled with colored patterns. These are held in a Graphics Presentation Space.

Dialog Data
: which includes items which the user can Select or Type into as part of a structured dialog between application and user. This is held in a Menu or Dialog Box. This data is only displayed on the screen and not on other devices such as printers since it relates directly to the user interface.

Each type of output data has its own set of functions for creating and modifying the data. The data is essentially independent of the place where it is eventually drawn - it is a logical representation of what is required. Thus, for example, a picture can be created in a Graphics Presentation Space, first drawn onto the screen and then drawn onto paper by a printer.

An application may have many instances of each type of data, depending upon the application's requirements. For example, an application would have many Menus and Dialog Boxes if it needed a lot of structured input from the user.

2.1.2.1.2 Devices

Output data is drawn onto a *Device.* For Dialog Data, the device is always the Screen. For Graphics and Alphanumerics data the application must *Associate* the Presentation Space with a Device. In these cases, the Device may be the Screen, a memory Bitmap, a Printer, a Plotter or a File. The association can be changed by the application so that the same data can be directed to a number of places in sequence, typically following user requests.

A device is logically represented by a *Device Context,* which encompasses the *Device Driver* required to use the device, and *State Data* which includes appropriate physical realizations of device dependent objects such as Text Fonts.

2.1.2.1.3 Screen Windows

Output data is drawn onto the Screen through one or more *Windows*. Each separate Presentation Space or Dialog Box is normally shown in its own window. The windows are created by the application. Windows are rectangular in shape and are fitted onto the screen in a *'Messy Desk'* arrangement. This means that the windows are treated like a series of rectangular pieces of paper on a desk. The windows can overlap one another. Where they overlap, only one of the windows can be seen - the windows have an ordering where one window lies 'on top' of another. Where they overlap, the 'topmost' is seen.

The screen displays all the currently visible windows of all the applications that are running in the Presentation Manager screen group. An application can control the ordering of its own windows relative to one another. It does not control the ordering of its windows relative to the windows of other applications. This is done by the Presentation Manager system according to user requests.

Windows are not used on devices other than the screen. This is because their main use is in enhancing the end-user interface. The screen is special - it is used in a highly interactive way and space for the display of important information is limited. Multiple applications can use other output devices serially, but in a highly interactive environment it is important for the user to be able to see and use multiple applications simultaneously. Similarly, objects such as Menus and Dialogs are used for short periods at a time and should not occupy screen space except when needed. Thus they are placed in windows which can be made invisible.

As well as being a place where an application can display data on the screen, windows have a User Interface aspect as well. The user can alter the position and/or the size of some (but not all) windows. This allows the user to layout work on the screen in a convenient way. To achieve this function, windows have a variety of *Controls* which occupy their borders and allow the user to manipulate the window in a number of ways.

2.1.2.2 Input Fundamentals

2.1.2.2.1 User Input

The end user of the system creates input using the Mouse and Keyboard devices. The user can create the following Input Events:

- Mouse Button up/down
- Mouse Movement
- Keyboard Key up/down

Each input event is called a *Message.* User input is asynchronous to applications - that is, the user can press keys or move the mouse to create input independently of the speed with which an application can process the input. All the user's input is buffered as a sequence of Messages in an *Input Queue* before reaching the application. This ensures that input is not lost and is correctly sequenced. The application reads the input messages from the queue one at a time using the *WinGet Message* or *WinPeek Message* functions.

There is a close relationship between user input and windows. The user directs input to one window at a time. Each input event is tagged with the ID of the window to which it is directed. Every window is associated with one input queue. Thus input related to a particular window can only be received by reading a particular queue. A single queue can receive input for any number of windows, however.

2.1.2.2.2 Other Kinds of Input

Input other than Mouse and Keyboard messages can also appear on an input queue. This includes:

- Timer messages, which occur after an application-set Timer expires.
- System messages, which inform the application of various system related events. A typical system message is the Paint message, which informs the application that a window (or part of a window) needs to be repainted/redrawn. This often occurs when some or all of the window becomes visible as a result of the user performing a windowing operation.
- Inter-application messages, which are sent from one application to another. These typically occur between applications which are cooperating in some way. Such messages have application-defined meaning.

2.1.3 Presentation Space, Device Contexts and Windows

2.1.3.1 Presentation Spaces

A presentation space contains the device-independent quantities required to perform output to an individual window or device. These include:

- A definition of the picture data itself.

  For VIO output, this is the VIO buffer. For a graphics picture, this could be a graphics segment store (though if non-retained processing is being used, segments are not kept by the system).
- Clipping region as defined by the application.
- Definition of any fonts required for drawing.

  This is essentially a logical description of the fonts, and does not include any physical font definitions.
- Coordinate mapping.

  An indication of how world coordinates are to be mapped to the device.
- A definition of the colors an application would like.
- The default attributes associated with the picture.

A presentation space is always required whenever the application wishes to use any of the GPI or Advanced Vio functions to output data on an output device or into a bitmap. All of the GPI and Advanced Vio calls require the presentation space handle to be specified as a parameter. The presentation space is created by the VioCreatePS or GpiCreatePS functions.

Before a presentation space can be used to draw a picture, it must be associated with a Device Context. (Refer to following section on Device Contexts.) After this has been done, any drawing operations issued to the presentation space cause output to occur on the device defined by the Device Context.

The presentation space can subsequently be associated with a different Device Context, and the picture redrawn on that device. Because all of the 'application intent' information is kept in the presentation space, the system is able to draw the picture as faithfully as possible on this second device.

Thus a picture which is currently visible on the screen can be printed by temporarily reassociating its presentation space with a Device Context whose device is a printer, and redrawing the presentation space. In order to continue drawing on the screen, the presentation space is now reassociated back to the screen Device Context.

(Note that the above scenario is only as simple if the entire picture definition has been retained in the presentation space. If non-retained graphics have been used, then the application needs to redraw the picture again after associating with the printer Device Context. However, it still does not need to respecify any of the logical objects, for example fonts, which it needs, since these are still kept in the presentation space.)

2.1.3.2 Device Contexts

A Device Context is the means of drawing to a particular device. It includes a device driver, and also physical realizations, where appropriate, of device-dependent objects which the drawing process requires.

There are four kinds of Device Context, as follows:

- Screen Device Context. This causes drawing to be performed to a particular window on the screen.
- Memory Device Context. This is used only for drawing to a memory bitmap.
- Metafile Device Context. This causes the picture to be transmitted to a metafile, which may be used to store a picture in editable form.
- Queued-device Device Context. This is used for some device other than the screen, for example, an attached printer or plotter, where the output is to go via the spooler.
- Directly attached-device Device Context. This is used for some device other than the screen, for example, an attached printer or plotter, where the output is not to go via the spooler.
- Information Device Context. This is used for some device other than the screen, for example, an attached printer or plotter, but where no output will occur. Its purpose is to satisfy queries.

A Device Context is required whenever the application wishes to use any of the GPI or Advanced Vio output functions. However, the Device Context is not normally specified on the GPI or Advanced Vio calls; instead a Device Context is associated with a particular presentation space, by the application issuing a GpiAssociate or a VioAssociate call (an implicit association is also possible in the Gpi case). The Device Context must be specifically created by the application in all cases. The application uses the DevOpenDC call to create a Device Context for a printer, bitmap, or metafile. In the case of the display screen, the Device Context for a window is created by a call to WinOpenWindowDC after the application creates the window with a WinCreateWindow call.

2.1.3.3 Windows

A Presentation Manager window is a rectangular area on the screen. A window contains visual data displayed from a Presentation Space, for example a Graphics (GPI) Presentation Space, which is associated with the window. Alternatively, a window could display data from a dialog box.

The screen can have many windows displayed and these may overlap.

Where overlap occurs, the windows have a priority ordering. At any point on the screen, the window with the highest priority gets displayed.

Presentation Manager windows are of two types:

Main Windows

A Main window has the property of being positioned relative to the screen itself. It is not related to any other window. Operations on one main window do not affect other main windows.

An application can have as many Main windows as it wants, within overall implementation limits.

A main window may be considered to be a child window of the entire screen.

Child Windows

A Child window has the property of being positioned relative to another window, termed its Parent. The Parent window can have multiple Children. Operations on the Parent window affect the Child. For example, moving the Parent moves the Child and hiding the Parent hides the Child.

A Child is constrained to fit within the client area of its Parent. A Child window always has a higher priority than its Parent, i.e., it cannot be hidden simply by virtue of being underneath its parent. A Child window may have Children of its own.

An application can have as many Child windows as it wants, within overall implementation limits.

2.1.4 Presentation Manager Functions

2.1.4.1 Output via GPI or Advanced Vio Functions

The data displayed in each window is held in a Presentation Space which is created and manipulated by the application separately from the window. The Presentation Space is different for different types of data - a GPI presentation space for Graphics/Image data, and an Advanced Vio presentation space for alphanumeric data.

The application output does not go directly from the presentation space to the screen window. It goes through a 'Device Context'. The Device Context encapsulates various physical characteristics of the output device. The main utility of the Device Context is for the support of other devices such as printers or memory bitmaps.

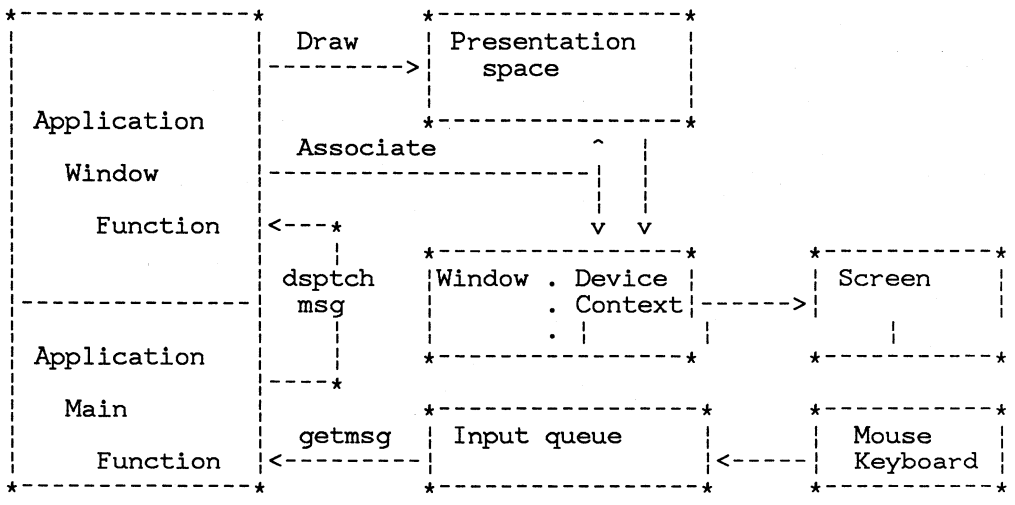


Figure 2.2 Application Model for Graphics and Alphanumerics

Thus to display some data on the screen, the application:

1. creates a Window (this will implicitly create a Device Context that is associated with the window)
2. creates a Presentation Space
3. associates the Presentation Space with the Device Context, so that data drawn from the Presentation Space goes into the Window
4. creates the data to display by operating on the Presentation Space
5. puts the data into the Window by means of a draw operation on the Presentation Space

Note that for GPI, items 2 and 3 can be combined in a single call, and also in non-retained mode items 4 and 5 are combined.

2.1.4.2 Output via User Controls Functions

When an application wishes to use the User Controls functions, it does not specifically create a presentation space. Instead, it interacts directly with a dialog box or menu object. These objects incorporate the concepts of window, presentation space, and Device Context.

In addition, the input that the application receives from the User Controls is processed by Presentation Manager and returned to the application in terms of the particular dialog box or menu.

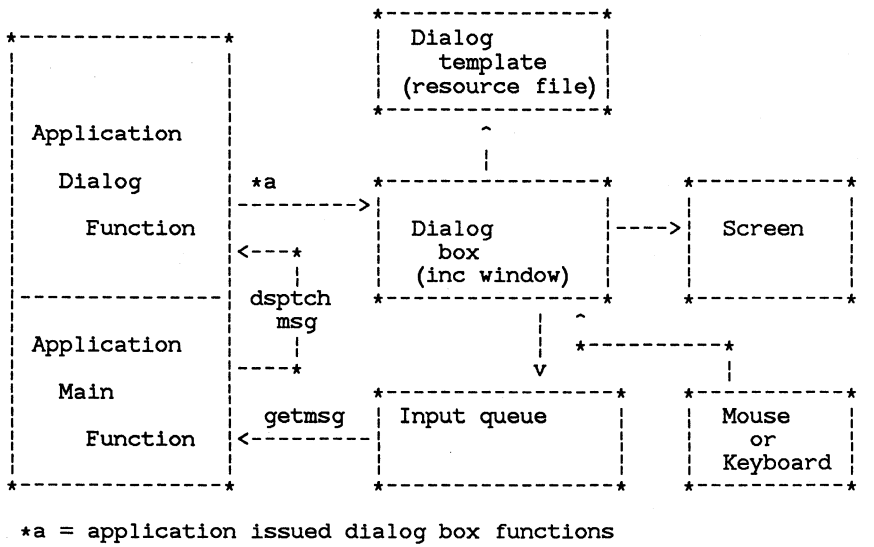


Figure 2.3 Presentation Manager Application Model for Dialog Boxes

2.1.4.3 Input Functions

In Presentation Manager, input to an application is a sequence of Messages generated by any of a number of sources. The Messages are placed on an application Input Queue in time order and are read from the queue by the application. All Presentation Manager applications have at least one input queue - the Default Input Queue. In addition, the application may create additional input queues. Every window created by the application will have a single input queue associated with it. Most input messages are associated with one of the application's windows, and are sent to the appropriate queue.

In addition to an input queue, every window normally has a window processing function associated with it. It is the job of the window processing function to process the input associated with a window. The main program of the application will read the input from the input queue by use of the WinGetMessage function, and then route the input to the appropriate window processing function by means of the WinDispatchMessage function.

It is the responsibility of the application to be 'well-behaved'. This means that an application must always issue a WinGetMessage to read its input queue within a short time period (e.g., 0.1 sec) of receiving the previous input. In order to achieve this, it must have dispatched the window processing function, and the window processing function must have completed and returned within the specified time.

If the application does not meet these requirements, various system functions and/or other applications will be locked out until the application completes processing and reissues its read on the input queue.

2.1.5 Sample Programs

A number of Sample Programs are supplied with Presentation Manager to help programmers understand the Presentation Manager API and give hints as to how the API can be used to achieve results. Each sample program tackles its own functional area:

- Use of Windows.
- Use of Menus and Dialogs.
- Advanced Vio alphanumerics for display of Text.
- Keyboard and mouse input.
- Graphics:
  - Direct (non-retained) drawing of pictures
  - Stored creation, drawing and editing of pictures
  - Correlation
  - Dragging
  - Bitmap operations
  - Printing
  - Typographic fonts
- Use of the clipboard.

2.1.6 Application Model

This section covers various aspects of writing an application to use the Presentation Manager facilities. Presentation Manager places a number of requirements on the way an application is structured and the way in which it uses certain facilities, especially Input and the Screen.

Note that special considerations apply to MS OS/2 applications which use multiprogramming methods, i.e., multiple processes or multiple execution threads.

2.1.6.1 Basic Application Structure

To use the Presentation Manager API, a program must call the *WinInitialize* function before any other Presentation Manager function. This function identifies the application to Presentation Manager and initializes the application's environment.

The WinInitialize function returns an *Anchor Block Handle* which has the purpose of holding the application's Presentation Manager environment data. The Anchor Block Handle must be stored by the application for later use - it is required by a number of Presentation Manager functions, such as WinCreateWindow.

Once WinInitialize has been called, the application can use any other Presentation Manager function - to display data on the screen or receive input from the user, for example. If another Presentation Manager function is called before WinInitialize, it fails and returns an error code.

When the application is about to finish, it should call the *WinTerminate* function. This is the inverse function to WinInitialize - it destroys the application's Presentation Manager environment. After WinTerminate has been called, the application cannot make any further Presentation Manager function calls. If the application makes any Presentation Manager function calls after WinTerminate has been called, the function fails and an error code is returned.

WinTerminate deallocates and destroys any Presentation Manager resources that were allocated to the application, such as Windows and Presentation Spaces. However, it is recommended that such resources are explicitly destroyed by the application before calling WinTerminate - this allows the application to perform a tidier 'cleaning up' of the resources, including saving data in disk files if required.

```
Program Fred;
                          -¦
****                       ¦
****                       >- Program Initialization
****                       ¦
                          -+

WinInitialize( );         --- Presentation Manager Initialization


                          -¦
****                       ¦
****                       >- Main body of the program.
****                       ¦
                          -+
```

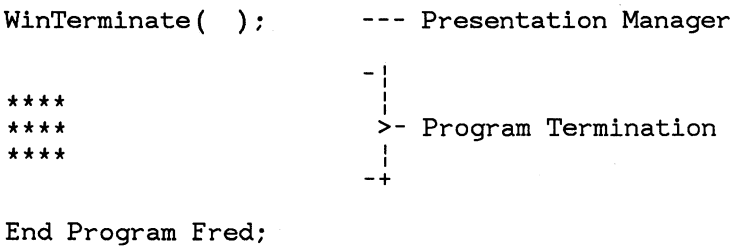


Figure 2.4 Application Structure

2.1.6.1.1 Normal and Abnormal Application Completion

An application which uses the Presentation Manager API normally allocates various resources to itself, such as Windows, Presentation Spaces and Input Queues. It is recommended that the application deallocates and destroys all these resources before it finishes.

If the application fails to deallocate any Presentation Manager resources before finishing, for example by failing to call WinTerminate, then these resources are still deallocated by the system. This occurs, in MS OS/2 terms, when the Exit List processing occurs. *Note:* This applies whether the application finishes normally or abnormally (due to some error). The Presentation Manager system ensures that no resources are left 'lying about' once the application finishes.

It is better if the application explicitly destroys any resources since the application can do things in a more coherent order - especially from the appearance of things on the screen. The application can also save away any data associated with the resources, if necessary.

2.1.6.2 The System Environment—The Shell and Other Applications

In Presentation Manager, an application does not stand on its own. It is part of a system which interfaces to the user. In particular, the User Interface Shell forms a major part of the interface and it is through the User Interface Shell that the application is started and is accessed when running.

2.1.6.2.1 Starting an Application

When the user starts an application, this is done via some selection(s) from windows in the User Interface Shell. The application is represented there by a *Long Name,* which is more meaningful to the user than an eight letter filename.

Once an application is running and it creates and displays a main window, the window must be given a Title, so that the user can identify the application.

The User Interface Shell also has a Switch List containing the names of all the main windows of applications in the system. This allows the user to find an application of interest when the system is running more than one application on the screen and some of the applications are obscured by other applications' windows. *It is the application's responsibility to put its entry into the Switch List.*

The Long Name by which an application is started can be found by calling the *WinGetStartupName* function, which is part of the Shell API. It is recommended that an application uses this name for its main window and for its entry in the Switch List. However, where the application is working with a particular data file, it is also recommended that the file name is appended to the Long Name to form the Window Title and the Switch List entry.

2.1.6.2.2 Main Window Title

The title of the application's main window is set in the *WinCreateFrameWindow* function when the window is created. However, the title can be updated subsequently by sending a *WM_SETWINDOWPARAMS* message to the window, for example if the application starts work on another file.

2.1.6.2.3 Switch List Entry Name

The Switch List entry is created by the *WinAddSwitchEntry* function. The Name displayed in the Switch List is specified as a parameter to this function, along with the Window Handle of the application's main window. When the user selects the Switch List entry belonging to the application, the main window is made Active and it is brought to the top of the stack of windows.

2.1.6.3 Program Structure and Windows— Window Procedures

A Screen Window is not only used as a place to put display data. Windows have an important role in a number of aspects of the Presentation Manager API:

- Display of multiple sets of data on the screen.
- Efficient use of scarce screen area.
- Handling of Input - both from the user and the system.
- Program structuring and partitioning.
- A seamless way of extending System functions.

2.1.6.4 Application Rules and Conventions

2.1.6.4.1 Mouse Button Activation of a Window

It is the application's job to transfer active status to one of its windows if it gets a mouse down message. It should do this by calling WinSetFocus or WinSetActiveWindow with the window that the mouse message was sent to.

2.1.6.4.2 Active Windows, Dialog Boxes and User Expectations

The application should not call WinSetActiveWindow arbitrarily to set the active status to one of its windows. This should only be done as the result of an explicit user action requesting a new window to become the active one or as the result of a message from the shell to the same effect.

Neither should an application display a dialog box or message box arbitrarily if it needs to tell the user something and it doesn't own the active window. Applications that need to do this should call DosBeep a few times and then call WinFlashWindow. WinFlashWindow will start the frame of the window flashing. The user will hear a beep and see the window flashing. The user can then choose to pay some attention to the application, and request that it become the active one, say by clicking the mouse with the pointer in the flashing window. The application can then call WinFlashWindow again to turn off the flashing, and bring up an appropriate Message or Dialog.

2.1.6.4.3 Mouse Pointer Shape Within a Window

It is the application's job to set the shape of the mouse cursor when it gets a WM_MOUSEMOVE message, using the WinSetPointer call. Child windows should send the WM_CONTROLPOINTER message to their parents, so their parents can have the choice of setting their cursor shapes. See 'Control Manager'.

2.1.6.5 Building an MS OS/2 Presentation Manager Application

This section describes the method of building an MS OS/2 Presentation Manager application. This includes details specific to MS OS/2 Presentation Manager applications that do not apply to other MS OS/2 applications. The reader is assumed proficient in building general MS OS/2 applications.

The following describes the source files required for MS OS/2 Presentation Manager, and the processes through which these are turned into an executable file. The application programmer is responsible for providing three types of source files:

- a resource file
- one or more source code files (i.e., C or assembler files)
- an MS OS/2 module definition file

2.1.6.5.1 Resource Files

The resource file contains descriptions of the application's user interface data, such as dialog boxes or menus. The application programmer defines these either through a text description, or by using a tool such as the dialog editor which will in turn create the text description.

The Resource Compiler understands these descriptions, and performs two functions in building an application. First, it compiles the text description into a binary format suitable for the MS OS/2 Presentation Manager system. Second, it inserts these binary resources into the executable file. The insertion must be done after linking the objects, i.e., the sequence is:

```
link
rc
```

The resource compiler is invoked through the command:

```
rc resourcefilename [exefilename]
```

The resourcefilename is the MS OS/2 filename of the resource text file. If no extension is specified, the extension is assumed to be RC. The exefilename is optional, and is the name of executable to insert the binary resources into. If it is not specified, then the default is the executable with the same filename as resourcefilename, i.e.:

```
rc sample
rc sample.rc
rc sample.rc sample.exe
```

would all compile the resources described in sample.rc and would insert them in sample.exe.

Compilation of the resources takes time, and the resources must be reinserted in the executable every time the application is relinked. Thus, to save application build time, the Resource Compiler has an option to compile the resources and then create an intermediate object file. This resource object file can then in turn be specified as input to the Resource Compiler, to complete the final step of insertion into the executable.

To create the intermediate object file, specify the "-r" option, which will create a file whose extension is RES.

Example:

```
rc -r sample.rc
link
rc sample.res sample.exe
```

2.1.6.5.2 Source Code

High level language files are compiled using the appropriate language compiler; assembler files must be assembled. In both cases, intermediate object files (.OBJ) should be created.

2.1.6.5.3 Module Definition File

All external entry points in an MS OS/2 Presentation Manager application must be EXPORTed in the Module Definition File (.DEF). See "Building an MS OS/2 Application" for a further description of the .DEF file.

2.1.6.5.4 Linking an MS OS/2 Presentation Manager Application

At link time, the developer must specify:

- the code object files to be linked (.OBJ's)
- the Module Definition File (.DEF)
- Libraries (.LIB's)

In order to resolve references to MS OS/2 Presentation Manager API, the developer must specify Wincalls.lib in addition to any other necessary libraries.

Sample Build Sequence

```
rc -r sample  => creates sample.res

Compile sample.c = creates sample.obj

Link sample.obj, sample.def, wincalls.lib => creates sample.exe

rc sample.res => modifies sample.exe

Sample.exe is now ready to run under MS OS/2 Presentation Manager.
```

2.1.7 Non-Reentrant Language Support

This section describes how to write a Presentation Manager application in a language which does not support either reentrancy or recursion into the application via callback functions. COBOL and FORTRAN are examples of languages which do not allow these features. Note that to fully appreciate the content of this section, it may be necessary to read and understand some of the detailed sections concerning the Presentation Manager API.

Some features of Presentation Manager are implemented with window procedures, which are reentrant application procedures. These features of Presentation Manager may not be available to an application written in a language not supporting such constructs. However, most features are available, but some must be implemented in a slightly different way.

The first step in writing a Presentation Manager application that does not use standard window procedures is to register a window class with a NULL window procedure address. This window class may be used withWinCreateStdWindow to create a client window.

The key difference between a no-window-proc window and a standard window is that messages may not be sent to the window. Since the frame window and the frame controls are of preregistered classes, these windows have window procedures, and thus may be sent messages.

The other difference is that WinDispatchMsg may not be used to dispatch queued messages to the window. Instead, messages must be processed directly as it is obtained with WinGetMsg or WinPeekMsg in the main loop. Messages for no-window-proc windows must be handled directly, but messages for other windows (such as the frame window or the frame controls) must be dispatched as usual with WinDispatchMsg.

Below is a simple skeletal application showing how a no-window-proc application might be structured. Notice that in this example, we check to see if a queue message is destined for the no-window-proc window, and if so, we call an application "window proc", which preserves much of the object-oriented flavor of the system.

```
/*
 * Register a no-window-proc class
 */
WinRegisterClass("MyClass", NULL, 0L, 0, NULL)

/*
 * Create a standard

hwndFrame = WinCreateStdWindow(NULL, FS_STANDARD, "MyClass",
        "Hello World", WS_VISIBLE, NULL, 0, &hwndMyWindow);

/*
 * Process messages from the queue
 */
while (WinGetMsg(hab, &qmsg, NULL, 0, 0)) {
    /*
     * If the messages is for my no-window-proc window,
     * then "dispatch" it directly to the "window procedure".
     * Since the "window proc" is not a true window proc, it need
     * not be declared FAR PASCAL or anything like that.
     In addition,
     * only queued messages will be received by the procedure.
     *
     * If not for no-window-proc window, dispatch the
     message as usual.
     */
    if (qmsg.hwnd == hwndMyWindow) {
        MyWindowProc(qmsg.hwnd, qmsg.msg, qmsg.mp1,
        qmsg.mp2);
    } else {
        WinDispatchMsg(&qmsg);
    }
}


/*
 * Here is an example "window procedure" for the no-window-proc
 * window. Again, this isn't a real window procedure, simply
 * a single place in the application where all messages for the
 * window go to.
 */
```

```
MyWindowProc(hwnd, msg, mp1, mp2)
HWND hwnd;
USHORT msg;
ULONG mp1;
ULONG mp2;
{
    switch (msg) {
    case WM_CHAR:
        break;
    case WM_COMMAND:
        break;
    case WM_PAINT:
        break;
    break;
}
```

Creating instances of any of the preregistered classes such as scroll bars or pushbuttons is no problem. However, not all of the features of controls may be available: any messages sent by the control can not be processed. However, messages POSTED by controls can be processed as usual in the main loop of the application.

Certain standard window messages are sent to the window, and thus may not be processed by no-window-proc applications. Many of these messages simply provide information that can be obtained by other means, and thus are not required. Other messages that are normally sent to a window are automatically posted in the queue instead of being sent to the window. This is called "message postback". Some messages are always processed in a default way, rather than being posted back.

Here are some guidelines regarding whether a message can be processed by a no-window-proc application:

1. All posted messages can be processed in the main loop.
2. Messages that reflect a state change that can be polled when necessary (e.g., WM_SIZE) are not posted back.
3. Messages that reflect state changes that cannot or are inconvenient to obtain by other means are posted back (WM_ACTIVATE, WM_ACTIVATETHREAD).
4. Messages that include far pointers to structures or other transient information must not be posted back. Only the default processing of the message is available.

Here are some of the restrictions on a no-window-proc application:

1. Only standard PC/UIS window size/move tracking, sizing, rearrangement is available.
2. No notification is made to the application when a window is created, destroyed, moved, sized, hidden or shown. Since generally only the application can make these state changes, this is not

really a problem. There are a number of ways to handle these situations:

- After the call is made that would normally send a notification (e.g., WinShowWindow), the application code that would correspond to the processing of the notification message can be called immediately after the call.
- In code that depends on a particular window state, such as the size of a window, the current state of the window can be compared with a saved copy of the previous state. If any changes are detected at this time, the application code to process the state change may be called.

3. Dialog box manager

   Not all features of the dialog box manager and dialog controls are available. Dialog boxes should be created with WinLoadDlg or WinCreateDlg, then initialized with in-line code (in place of processing the WM_INITDLG message). WinProcessDlg must then be called to handle user input for the dialog. When WinProcessDlg returns, the field values of the dialog box may be queried, and the window destroyed.

   Note that the restrictions apply both for modal and for modeless dialogs. Modal dialogs are particularly problematic in that none of the messages relating to the dialog can be 'seen' by the application during the user's interaction with the dialog.

   The dialog procedure of the WC_DIALOG window class operates so that if there is no application dialog procedure, the default dialog proc handles WM_COMMAND messages by issuing WinDismissDlg and passing back the command value as a parameter.

   In other words, the winprocless application looks like:

```
hwnd = WinLoadDlg(..);
code = WinProcessDlg(...);
switch (code) {
case DID_OK:
  /* user pressed the OK button, do whatever's necessary */
  break;
case DID_CANCEL:
  /* user pressed the CANCEL button, do whatever's necessary */
  break;
}
WinDestroyWindow(hwnd);
```

   An important point to note is that the default processing of WM_COMMAND messages means that all pushbuttons cause WinProcessDlg to return. If the application needs to go back into the dialog, WinProcessDlg must be called again.

4. Controls:

   Only notifications from controls that are posted may be used. For

example, pushbuttons post WM_COMMAND messages, and scroll bars post WM_H/VSCROLL messages, which may be obtained by calling WinGetMsg or WinPeekMsg. Other notifications, such as WM_INITMENU, are sent, and are thus not available. Most of the common features of the standard controls are available.

All controls may be sent messages as usual.

Summary of control notifications available:

Menus:

- WM_COMMAND
- To implement WM_INITMENU, a mouseclick or key message to a menu control window can be recognized in the main loop, and the menu initialized before WinDispatchMsg is called to dispatch the message to the menu control.

Scroll bars:

- WM_HSCROLL
- WM_VSCROLL

Pushbuttons:

- WM_COMMAND

Edit controls:

- None.

Listboxes:

- None.

5. Clipboard Manager

   No delayed format rendering is possible with no-window-proc windows.

6. System color change notification

   Postback is used to post message to application. Application must ensure that all parts of the application are redrawn as necessary when the message is eventually received and processed in the main loop, since the message may not have been received until after all windows have been redrawn.

7. No WM_OTHERWINDOWDESTROYED destroy registration notification.

8. WM_SETFOCUS, WM_ACTIVATE notification messages are posted back. No-window-proc apps may not assume synchronous notification of these events. Also, the window handle parameters of these messages (hwndFocusPrev, hwndActivePrev, etc.) are NULL when posted back.

2.1.8 Background to User Interface

The Presentation Manager specification (the user interface), is a set of rules intended to provide end users with a consistent, easy-to-use interface across applications.

It includes many elements of user interaction with the system, such as menu selection and text string input, but it does not include interactions specific to applications, such as spreadsheet editing.

Where an application has user interaction in areas covered by user interface rules, it must conform to the user interface.

The principal topics in the user interface are as follows:

- Key assignments
- Menu colors
- Application action bars
- Pop-down menus
- Scroll bars
- Types of selection fields
- Entry fields
- Message and Help panels
- Window sizing and moving

MS OS/2 Presentation Manager allows all applications to conform to the user interface. For some rules, MS OS/2 Presentation Manager enforces conformance by taking over complete parts of the operator interaction. Thus, for these interactions, the only way the application could avoid being in conformance would be to rewrite part of the code provided with MS OS/2 Presentation Manager.

2.1.9 Naming Conventions

Here is a short description of the variable & argument naming conventions used in the Presentation Manager spec. A name is made up of a tag prefix and an optional identifier. The tag is all lower case, and the identifier begins with an upper case letter. You can either make up your own tags for new data types, or use some combination of the standard tags.

2.1.9.1 Constant Names

All constants are written in upper case. If applicable, constant names have a prefix derived from the name of a function, message, or idea associated with the constant. For example:

```
WM_CREATE           - Window message (WM_*)
WS_CLIPSIBLINGS     - Window style (WS_*)
DT_CENTER           - DrawText() code (DT_*)
```

2.1.9.2 Type Names

Type names are written in upper case. Type names are usually longer and more descriptive than their variable and argument prefixes; for example:

```
Type                Prefix

WRECT               rc
WPOINT              pt
```

2.1.9.3 Variable and Argument Names

A name is made up of a tag prefix and an optional identifier. The tag is all lower case, and the identifier begins with an upper case letter. You can either make up your own tags for new data types, or use some combination of the standard tags.

```
Standard name prefixes:

    p       - near pointer
    lp      - far pointer
    d       - delta
    c       - count
    i       - index
    rg      - array

    f       - boolean
    h       - handle
    ch      - character
    b       - byte
    w       - word
    l       - long

    id      - ID
    it      - item
    cmd     - command

    pfn     - near function address
    lpfn    - far function address
    psz     - near ptr to zero terminated string
    lpsz    - far ptr to zero terminated string
```

```
    fs      - 16-bit packed array of flags/bits
    lrgf    - 32-bit packed array of flags/bits
    brgf    - 8-bit packed array of flags/bits

Standard type abbreviations:

    hab     - Anchor block handle

    hwnd    - window handle
    rc      - rectangle
    pt      - point
    hmenu   - menu handle
    t       - 32-bit millisecond value
    x       - x coordinate
    y       - y coordinate

    hps     - PS handle

    hvps    - VIO PS handle

    hdc     - device context handle
    hbm     - bitmap handle
    hrgn    - region handle
    hptr    - pointer handle
    hdc     - DC handle

    msg     - window message ID
    style   - 32-bit window style

Standard type identifiers

    Next    - Next
    Prev    - Previous
    First   - First value (used with Last)
    Last    - Last value (== last value, not one greater)
    Min     - Minimum value (used with Max)
    Max     - Maximum value (one past last possible)

Examples:

    pch             - Near pointer to a character (or characters)
    rgbBuffer       - Array of bytes
    dx              - Delta-x value
    cyMax           - Max count of y coordinates
    rgfMenu         - Menu command values
    lpphwnd         - Far pointer to a near pointer to a window handle.
```

2.1.9.4 Assembly-Language Structure Fields

In assembly language, all structure field names must be unique. Since not all structure fields have unique names, the assembly language convention is that all field names are prepended with the structure type abbreviation and an underscore. Here are some examples:

```
WRECT xLeft field: rc_xLeft
WPOINT y field: pt_y
```

2.1.9.5 Standard Data Types Used in This Document

Below is a list of the standard data types used in this document:

Type	Description
CHAR	Signed 8-bit value or character
SHORT	Signed 16-bit value
LONG	Signed 32-bit value'
UCHAR	Unsigned 8-bit value
USHORT	Unsigned 16-bit value
ULONG	Unsigned 32-bit value
PSZ	Far pointer to a zero terminated character string
PCH	Far pointer to a character string
BOOL	16-bit Boolean (zero => FALSE, non-zero => TRUE)
PFN	Far pointer to a procedure
PQMSG	Far pointer to a qmsg structure
PWRECT	Far pointer to a rect structure
PWPOINT	Far pointer to a point structure
PSWP	Far pointer to a window position structure
HWND	Window handle
HPROGRAM	Program Handle
HSWITCH	Switch List Handle
HPS	Presentation space handle
HENUM	Window enumeration handle (WinBegin/EndEnumWindows)

HMODULE
: Module handle

HAB
: Anchor block handle

HACCEL
: Accelerator table handle

HRGN
: Region handle

HBITMAP
: Bitmap handle

HPOINTER
: Pointer or icon handle

HDC
: Device Context handle

HHEAP
: Heap handle

HATOMTBL
: Atom table handle

HMQ
: Message queue handle

HSPL
: Spooler handle

HPROC
: Spool queue processor handle

Here are the standard declarations in C for these types:

```
typedef unsigned char UCHAR;

typedef unsigned short USHORT;

typedef short SHORT;

typedef unsigned long ULONG;

typedef char FAR *PCH;

typedef int BOOL;

typedef long LONG;

typedef int (FAR * PFN) ();
```

2.1.10 Return Code Conventions

This section documents the strategy for return codes used for Presentation Manager.

2.1.11 Error Conventions

There are six categories of procedure return values:

1. Restricted range return values

   An error is indicated by returning a value outside the specified range, or a special value withing the range. Any routines that return an HWND, for instance, fall into this category because a value of 0L (NULL) indicates an invalid window handle and thus an error. The "special" error value doesn't have to be 0. Note that all procedures that return a region code fall into this category, because they return RGN_ERROR when an error occurs.

2. Defaulted return values

   Reasonable default behavior exists if an error occurs. The procedures generally return 0 in the default or error case. However, since reasonable, documented default behavior is executed in an error condition, it is likely that applications will never explicitly test these return values for errors.

3. Failsafe unrestricted range return values

   There is no known error case, or it isn't efficient to test for one.

4. No return value.

   The function doesn't return any interesting information EXCEPT whether or not we succeeded (fSuccess Boolean).

5. Structure return value

   Information is returned by passing long pointer to a structure (which often contains input information as well). Returns fSuccess Boolean.

6. Window message return value

   The procedure return value is the same as the window message return value for the corresponding message.

```
category 1: Restricted Range Return Values (returns NULL, 0, or 0L if error)
=============================================
HWND FAR PASCAL WinCreateWindow(HWND, PSZ, PSZ, ULONG, SHORT, SHORT, SHORT,
                                SHORT, HWND, HWND, USHORT, PVOID,
                                PVOID)
HWND FAR PASCAL WinWindowFromID(HWND, USHORT);
HWND FAR PASCAL WinQueryWindow(HWND, SHORT, BOOL);
HWND FAR PASCAL WinLockWindow(HWND, BOOL);
```

```
HWND FAR PASCAL WinQueryActiveWindow(HWND, BOOL);
HWND FAR PASCAL WinQuerySysModalWindow(HWND, BOOL);
HWND FAR PASCAL WinGetNextWindow(HENUM);
HWND FAR PASCAL WinWindowFromPoint(HWND, PWPOINT, BOOL, BOOL);
HWND FAR PASCAL WinWindowFromDC(HDC);
HWND FAR PASCAL WinQueryCapture(HWND, BOOL);
HWND FAR PASCAL WinQueryDesktopWindow(HAB, HDC);
HWND FAR PASCAL WinQueryFocus(HWND, BOOL);
HWND FAR PASCAL WinLoadMenu(HWND, HMODULE, USHORT);
HWND FAR PASCAL WinCreateMenu(HWND, PVOID);
HWND FAR PASCAL WinLoadDlg(HWND, HWND, PFNWP, HMODULE, USHORT, PVOID);
HWND FAR PASCAL WinCreateDlg(HWND, HWND, PFNWP, PDLGT, PVOID);
HWND FAR PASCAL WinEnumDlgItem(HWND, HWND, USHORT, BOOL);
HWND FAR PASCAL WinCreateStdWindow(HWND, ULONG, PSZ, PSZ, ULONG,
                                   HMODULE, USHORT, PHWND FAR *);
HWND FAR PASCAL WinCreateStdWindowIndirect(HWND, ULONG, PSZ, PSZ, ULONG,
                                           BYTE FAR *, BYTE FAR *, HPOINTER,
                                           HWND FAR *);
HWND FAR PASCAL WinQueryClipbrdOwner(HAB, BOOL);
HWND FAR PASCAL WinQueryClipbrdViewer(HAB, BOOL);
HENUM FAR PASCAL WinBeginEnumWindows(HWND);
HMQ FAR PASCAL WinCreateMsgQueue(HAB, SHORT);
HACCEL FAR PASCAL WinLoadAccelTable(HAB, HMODULE, USHORT);
HACCEL FAR PASCAL WinCreateAccelTable(HAB, PACCELTABLE);
HACCEL FAR PASCAL WinQueryAccelTable(HAB, HWND);
HPOINTER FAR PASCAL WinLoadPointer(HWND, HMODULE, USHORT);
HPOINTER FAR PASCAL WinCreatePointer(HWND, HBITMAP, BOOL, SHORT, SHORT);
HPOINTER FAR PASCAL WinQuerySysPointer(HWND, SHORT, BOOL);
HPS FAR PASCAL WinBeginPaint(HWND, HPS, PWRECT);
HPS FAR PASCAL WinGetPS(HWND);
HDC FAR PASCAL WinOpenWindowDC(HWND);
PFNWP  FAR PASCAL WinSubclassWindow(HWND, PFNWP);
SHORT FAR PASCAL WinQueryUpdateRegion(HWND, HRGN);
SHORT FAR PASCAL WinExcludeUpdateRegion(HPS, HWND);
SHORT FAR PASCAL WinScrollWindow(HWND, SHORT, SHORT, PWRECT, PWRECT, HRGN,
                                 PWRECT, USHORT);
SHORT FAR PASCAL WinDrawText(HPS, SHORT, PSZ, PWRECT, ULONG, ULONG, USHORT);
USHORT FAR PASCAL WinProcessDlg(HWND);
USHORT FAR PASCAL WinDlgBox(HWND, HWND, PFNWP, HMODULE, USHORT, PVOID);
SHORT FAR PASCAL WinSubstituteStrings(HWND, PSZ, SHORT, PSZ);
ULONG FAR PASCAL WinQueryClipbrdData(HAB, USHORT);
USHORT FAR PASCAL WinEnumClipbrdFmts(HAB, USHORT);
SHORT FAR PASCAL WinLoadString(HAB, HMODULE, USHORT, SHORT, PSZ);
SHORT FAR PASCAL WinLoadMessage(HAB, HMODULE, USHORT, SHORT, PSZ);
USHORT FAR PASCAL WinStartTimer(HAB, HWND, USHORT, USHORT);
HPOINTER FAR PASCAL WinQueryPointer(HWND);


Returns MBID_ERROR
USHORT FAR PASCAL WinMessageBox(HWND, HWND, PSZ, PSZ, USHORT, USHORT);
category 2: Defaulted return values


=========================================
Subcategory A: Default error return is TRUE
BOOL FAR PASCAL WinIsRectEmpty(HAB, PWRECT);

Subcategory B: Default error return is FALSE
BOOL FAR PASCAL WinIntersectRect(HAB, PWRECT, PWRECT, PWRECT);
BOOL FAR PASCAL WinUnionRect(HAB, PWRECT, PWRECT, PWRECT);
BOOL FAR PASCAL WinSubtractRect(HAB, PWRECT, PWRECT, PWRECT);
```

```
BOOL FAR PASCAL WinEqualRect(HAB, PWRECT, PWRECT);
BOOL FAR PASCAL WinPtInRect(HAB, PWRECT, PWPOINT);
BOOL FAR PASCAL WinIsWindowEnabled(HWND);
BOOL FAR PASCAL WinIsWindowVisible(HWND);
BOOL FAR PASCAL WinIsWindow(HAB, HWND);
BOOL FAR PASCAL WinIsChild(HWND, HWND);
BOOL FAR PASCAL WinIsThreadActive(HAB);
BOOL FAR PASCAL WinInSendMsg(HAB);
BOOL FAR PASCAL WinPeekMsg(HAB, PQMSG, HWND, USHORT, USHORT, USHORT);
BOOL FAR PASCAL WinGetMsg(HAB, PQMSG, HWND, USHORT, USHORT);
BOOL FAR PASCAL WinTimeoutSendMsg(HWND, USHORT, ULONG, ULONG, ULONG,
                                  ULONG FAR *);
BOOL FAR PASCAL WinCallMsgFilter(HAB, PQMSG, SHORT);
BOOL FAR PASCAL WinTranslateAccel(HAB, HWND, HACCEL, PQMSG);
BOOL FAR PASCAL WinEnablePhysInput(HWND, BOOL);
BOOL FAR PASCAL WinTrackRect(HWND, HPS, PTRACKINFO);
BOOL FAR PASCAL WinProcessDlgMsg(HWND, PQMSG);
BOOL FAR PASCAL WinQueryClipbrdFmtInfo(HAB, USHORT, PUSHORT);

Subcategory C: Default error return is 0 or 0L
SHORT FAR PASCAL WinFormatFrame(HWND, PWRECT, PSWP, SHORT, PWRECT);
SHORT FAR PASCAL WinQueryWindowText(HWND, SHORT, PSZ);
SHORT FAR PASCAL WinMultWindowFromIDs(HWND, PHWND, USHORT, USHORT);
SHORT FAR PASCAL WinQueryClassName(HWND, SHORT, PSZ);
SHORT FAR PASCAL WinQueryWindowTextLength(HWND);
SHORT FAR PASCAL WinQueryWindowLockCount(HWND);
USHORT FAR PASCAL WinCopyAccelTable(HACCEL, PACCELTABLE, USHORT);
ULONG FAR PASCAL WinQueryQueueStatus(HWND);
SHORT FAR PASCAL WinQueryKeyState(HWND, SHORT);
SHORT FAR PASCAL WinQueryPhysKeyState(HWND, SHORT);

category 3: Failsafe unrestricted return values (returns 0 or 0L if error)
=========================================
ULONG FAR PASCAL WinGetCurrentTime(HAB);
USHORT FAR PASCAL WinQuerySysValue(HWND, SHORT);
ULONG FAR PASCAL WinQuerySysColor(HAB, SHORT);
ULONG FAR PASCAL WinQueryMsgTime(HAB);
USHORT FAR PASCAL WinQueryWindowUShort(HWND, SHORT);
ULONG FAR PASCAL WinQueryWindowULong(HWND, SHORT);
USHORT FAR PASCAL WinQueryVersion (HAB)

category 4: No return value (fSuccess return)
=======================================
USHORT FAR PASCAL WinQueryWindowProcess(HWND, PUSHORT, PUSHORT);
BOOL FAR PASCAL WinFillRect(HPS, PWRECT, COLOR);
BOOL FAR PASCAL WinRegisterClass(HAB, PSZ, PFNWP, ULONG, USHORT, HMODULE);
BOOL FAR PASCAL WinDestroyWindow(HWND);
BOOL FAR PASCAL WinEnableWindow(HWND, BOOL);
BOOL FAR PASCAL WinShowWindow(HWND, BOOL);
BOOL FAR PASCAL WinEnableWindowUpdate(HWND, BOOL);
BOOL FAR PASCAL WinSetWindowText(HWND, PSZ);
SHORT FAR PASCAL WinQueryWindowLockCount(HWND);
BOOL FAR PASCAL WinSetWindowPos(HWND, HWND, SHORT, SHORT, SHORT, SHORT,
                                USHORT)
BOOL FAR PASCAL WinSetMultWindowPos(HAB, PSWP, SHORT);
BOOL FAR PASCAL WinSetActiveWindow(HWND, HWND, BOOL);
BOOL FAR PASCAL WinFlashWindow(HWND, BOOL);
BOOL FAR PASCAL WinSetSysModalWindow(HWND, HWND);
BOOL FAR PASCAL WinSetWindowUShort(HWND, SHORT, USHORT);
BOOL FAR PASCAL WinSetWindowULong(HWND, SHORT, ULONG);
BOOL FAR PASCAL WinEndEnumWindows(HENUM);
BOOL FAR PASCAL WinDestroyMsgQueue(HMQ);
```

```
BOOL FAR PASCAL WinWaitMsg(HAB, USHORT, USHORT);
BOOL FAR PASCAL WinPostMsg(HWND, USHORT, MPARAM, MPARAM);
BOOL FAR PASCAL WinPostQueueMsg(HMQ, USHORT, MPARAM, MPARAM);
BOOL FAR PASCAL WinSetMsgInterest(HWND, USHORT, SHORT);
BOOL FAR PASCAL WinDestroyAccelTable(HACCEL);
BOOL FAR PASCAL WinSetAccelTable(HAB, HWND, HACCEL);
BOOL FAR PASCAL WinSetKeyboardStateTable(HWND, PBYTE, BOOL);
BOOL FAR PASCAL WinSetSysValue(HWND, SHORT, LONG);
BOOL FAR PASCAL WinSetSysColors(HWND, SHORT, PSHORT, PULONG);
BOOL FAR PASCAL WinEndPaint(HPS);
BOOL FAR PASCAL WinReleasePS(HPS);
BOOL FAR PASCAL WinInvalidateRect(HWND, PWRECT, BOOL);
BOOL FAR PASCAL WinInvalidateRegion(HWND, HRGN, BOOL);
BOOL FAR PASCAL WinValidateRect(HWND, PWRECT, BOOL);
BOOL FAR PASCAL WinValidateRegion(HWND, HRGN, BOOL);
BOOL FAR PASCAL WinUpdateWindow(HWND);
BOOL FAR PASCAL WinLockScreen(HWND, HWND);
BOOL FAR PASCAL WinLockVisRegions(HWND, BOOL);
BOOL FAR PASCAL WinDrawBitmap(HPS, HBITMAP, PWRECT, PWPOINT, ULONG,
                              ULONG, USHORT);
BOOL FAR PASCAL WinInvertRect(HPS, PWRECT);
BOOL FAR PASCAL WinShowTrackRect(HWND, BOOL);
BOOL FAR PASCAL WinDrawBorder(HPS, PWRECT, SHORT, SHORT, ULONG,
                              ULONG, USHORT);
BOOL FAR PASCAL WinDismissDlg(HWND, USHORT);
BOOL FAR PASCAL WinSetDlgItemShort(HWND, USHORT, USHORT, BOOL);
BOOL FAR PASCAL WinAlarm(HWND, USHORT);
BOOL FAR PASCAL WinCreateFrameControls(HWND, ULONG, PSZ, HMODULE);
BOOL FAR PASCAL WinClearMinPosition(HWND);
BOOL FAR PASCAL WinOpenClipbrd(HAB);
BOOL FAR PASCAL WinCloseClipbrd(HAB);
BOOL FAR PASCAL WinEmptyClipbrd(HAB);
BOOL FAR PASCAL WinSetClipbrdOwner(HAB, HWND);
BOOL FAR PASCAL WinSetClipbrdData(ULONG, USHORT, USHORT);
BOOL FAR PASCAL WinSetClipbrdViewer(HAB, HWND);
BOOL FAR PASCAL WinSetHook(HAB, HMQ, SHORT, PFN, HMODULE);
BOOL FAR PASCAL WinReleaseHook(HAB, HMQ, SHORT, PFN, HMODULE);
BOOL FAR PASCAL WinStopTimer(HAB, HWND, USHORT);
BOOL FAR PASCAL WinDestroyPointer(HPOINTER);
BOOL FAR PASCAL WinShowPointer(HWND, BOOL);
HPOINTER FAR PASCAL WinCreatePointer(HWND, HBITMAP, BOOL, SHORT, SHORT);
BOOL FAR PASCAL WinDestroyPointer(HPOINTER);
BOOL FAR PASCAL WinSetPointerPos(HWND, SHORT, SHORT);
BOOL FAR PASCAL WinDrawPointer(HPS, SHORT, SHORT, HPOINTER, USHORT);
USHORT FAR PASCAL WinRegisterWindowDestroy(HWND, BOOL);
BOOL FAR PASCAL WinSetParent(HWND, HWND, BOOL);
BOOL FAR PASCAL WinSetOwner(HWND, HWND);
BOOL FAR PASCAL WinSetPointer(HWND, HPOINTER);
BOOL FAR PASCAL WinSetCapture(HWND, HWND);
BOOL FAR PASCAL WinSetFocus(HWND, HWND);
BOOL FAR PASCAL WinShowPointer(HWND, BOOL);
MRESULT FAR PASCAL WinBroadcastMsg(HAB, USHORT, MPARAM, MPARAM, BOOL);

category 5: Structure return value (fSuccess return)
=============================================
BOOL FAR PASCAL WinSetRect(HAB, PWRECT, SHORT, SHORT, SHORT, SHORT);
BOOL FAR PASCAL WinCopyRect(HAB, PWRECT, PWRECT);
BOOL FAR PASCAL WinSetRectEmpty(HAB, PWRECT);
BOOL FAR PASCAL WinOffsetRect(HAB, PWRECT, SHORT, SHORT);
BOOL FAR PASCAL WinInflateRect(HAB, PWRECT, SHORT, SHORT);
BOOL FAR PASCAL WinQueryWindowRect(HWND, PWRECT);
BOOL FAR PASCAL WinQueryWindowPos(HWND, PSWP);
```

```
BOOL FAR PASCAL WinQueryClassInfo(HAB, PSZ, PCLASSINFO);
BOOL FAR PASCAL WinMapWindowPoints(HWND, HWND, PWPOINT, SHORT);
BOOL FAR PASCAL WinQueryMsgPos(HAB, PWPOINT);
BOOL FAR PASCAL WinQueryUpdateRect(HWND, PWRECT);
BOOL FAR PASCAL WinMapDlgPoints(HWND, PWPOINT, SHORT, BOOL);
BOOL FAR PASCAL WinCalcFrameRect(HWND, PWRECT, BOOL);
BOOL FAR PASCAL WinQueryPointerInfo(HPOINTER, PPOINTERINFO);
BOOL FAR PASCAL WinQueryPointerPos(HWND, PWPOINT);
BOOL FAR PASCAL WinQueryPointerInfo(HPOINTER, PPOINTERINFO);
BOOL FAR PASCAL WinMakeRect(HAB, PWRECT);
BOOL FAR PASCAL WinMakePoints(HAB, PWPOINT, SHORT);
BOOL FAR PASCAL WinGetMinPosition(HWND, PSWP, PWPOINT);
BOOL FAR PASCAL WinGetMaxPosition(HWND, PSWP);
BOOL FAR PASCAL WinGetInitialWindowPos(HWND, SWP FAR *);
BOOL FAR PASCAL WinQueryDlgItemShort(HWND, USHORT, PSHORT BOOL);

category 6: Message type specific return values
=============================================
MRESULT FAR PASCAL WinSendMsg(HWND, USHORT, MPARAM, MPARAM);
ULONG FAR PASCAL WinDispatchMsg(HAB, PQMSG);
MRESULT FAR PASCAL WinSendDlgItemMsg(HWND, USHORT, USHORT, MPARAM, MPARAM);
```

All Presentation Manager messages can be split into the following categories and subcategories:

1. Defaulted return values. FALSE, 0, or 0L is returned if an error occurs.
2. Restricted range return values. Specific values are returned if error occurs.
3. Notification messages sent by Presentation Manager with no return value. Sent app in a situation where processing of message is not necessary, and/or app cannot affect or prevent actions taken by the sender. (e.g., WM_DESTROY)
4. Messages that are posted, and thus have no return value (e.g., WM_QUIT)

```
Category 1: Defaulted return values:
============

fSuccess return values:

WM_SETWINDOWPARAMS
WM_QUERYWINDOWPARAMS
WM_SETICON
WM_SETICONSLOT
BM_SETDEFAULT
MM_QUERYITEM
MM_SETITEM
MM_SELECTITEM
MM_SETITEMHANDLE
MM_SETITEMTEXT
WM_FLASHWINDOW
EM_SETTEXTLIMIT
LM_SETITEMHANDLE
LM_SETITEMHEIGHT
EM_CUT
```

```
EM_COPY
EM_CLEAR
EM_PASTE
BM_SETHILITE
BM_SETCHECK
MM_SETITEMATTR
BM_CLICK
MM_DELETEITEM
MM_REMOVEITEM
MM_STARTMENUMODE
MM_ENDMENUMODE
SZM_TRACKSIZE
TBM_SETSTATE
TBM_TRACKMOVE
TBM_SHOWICONTEXT
SBM_SETSCROLLBAR
SBM_SETPOS
EM_SETSEL
LM_SETTOPINDEX
LM_DELETEITEM
LM_SELECTITEM
LM_SETITEMTEXT
WM_ACTIVATEFRAME
WM_SETFRAMEPOS

fProcessed return values:

WM_INITDLG
WM_SETACCELTABLE
WM_DRAWITEM
WM_QUERYTRACKINFO
WM_QUEUESYNC
WM_CHAR
WM_FORMATFRAME
WM_UPDATEFRAME
WM_ERASEBACKGROUND
WM_MAXIMIZE
WM_MINIMIZE
WM_MOUSEMOVE
WM_BUTTON1DOWN
WM_BUTTON1UP
WM_BUTTON1DBLCLK
WM_BUTTON2DOWN
WM_BUTTON2UP
WM_BUTTON2DBLCLK
WM_BUTTON3DOWN
WM_BUTTON3UP
WM_BUTTON3DBLCLK
WM_ACTIVATE
WM_RENDERALLFMTS

fError return value:

WM_CREATE

fValid return value:

WM_VALIDATEACCEL

fAdjusted return value:

WM_ADJUSTWINDOWPOS
```

```
Category 2: Restricted range return value
============

WM_RENDERFMT
WM_QUERYMINMAXINFO
WM_SUBSTITUTESTRING
WM_QUERYACCELTABLE
WM_QUERYDLGCODE
WM_NEXTOWNEDMENU
WM_QUERYICON
WM_QUERYICONSLOT
BM_QUERYCHECKINDEX
BM_QUERYCHECK
BM_QUERYHILITE
MM_INSERTITEM
MM_QUERYSELITEMID
MM_QUERYITEMTEXT
MM_QUERYITEMTEXTLENGTH
MM_ITEMPOSITIONFROMID
MM_ITEMIDFROMPOSITION
MM_QUERYITEMATTR
MM_QUERYITEMCOUNT
WM_MEASUREITEM
TBM_QUERYSTATE
SBM_QUERYPOS
SBM_QUERYRANGE
EM_QUERYCHANGED
EM_QUERYSEL
LM_QUERYITEMCOUNT
LM_INSERTITEM
LM_QUERYSELECTION
LM_QUERYITEMTEXTLENGTH
LM_QUERYITEMTEXT
LM_QUERYITEMHANDLE
LM_SEARCHSTRING
LM_QUERYTOPINDEX
WM_CONTROLPOINTER
WM_CONTROLHEAP
WM_HITTEST
WM_NULL

Category 3: Notification messages (always return 0L)
============
WM_DESTROY
WM_OTHERWINDOWDESTROYED
WM_ENABLE
WM_SHOW
WM_MOVE
WM_SIZE
WM_CALCVALIDRECTS
WM_ACTIVATETHREAD
WM_SETFOCUS
WM_CANCELMODE
WM_SYSCOLORCHANGE
WM_PAINT
WM_CONTROL
WM_INITMENU
WM_MENUSELECT
```

```
Category 4: Posted messages (always return OL)
================
WM_ERROR
WM_TIMER
WM_SEM1
WM_SEM2
WM_SEM3
WM_SEM4
WM_COMMAND
WM_SYSCOMMAND
WM_HELP
WM_HSCROLL
WM_VSCROLL
WM_QUIT
WM_DESTROYCLIPBOARD
WM_PAINTCLIPBOARD
WM_SIZECLIPBOARD
WM_HSCROLLCLIPBOARD
WM_VSCROLLCLIPBOARD
WM_DRAWCLIPBOARD
```

2.1.11.1 Error Severity

Errors fall into one of the following categories:

Warning
: The function detected a problem but took some remedial action which enabled the function to complete successfully.

  Note that a function which detects an error of "Warning" severity *does not return a function value corresponding to "error"*. This is because in most cases, there is valid data to return.

  Thus an application using a function which can generate errors with "Warning" severity level must use WinGetLastError to identify occasions on which such errors occur, if necessary.

Error
: The function detected a problem for which it could not take any sensible remedial action. The system will be able to recover from the problem, in the sense that the state of the system, with respect to the application remains the same as at the time when the function was requested, i.e., the system has not partially executed the function.

Severe Error
: The function detected a problem from which the system cannot reestablish its state, with respect to the application, at the time when that function was requested, i.e., the system has partially executed the function, and therefore necessitates the application performing some corrective activity in order to restore the system to some known state.

Unrecoverable Error
: The function detected some problem from which the system cannot reestablish its state, with respect to the application, at the

time when that call was issued and it is possible that the application cannot perform some corrective action in order to restore the system to some known state, e.g., the application provides the address of the anchor block which the system discovers is apparently corrupted.

Severity levels are 16 bit unsigned integers, with the following values:

```
SEVERITY_NOERROR                0x0000
SEVERITY_WARNING                0x0004
SEVERITY_ERROR                  0x0008
SEVERITY_SEVERE                 0x000C
SEVERITY_UNRECOVERABLE          0x0010
```

2.1.11.2 Error Codes

WinGetLastError returns a 32 bit value. The format of this value is:

```
High uint: 16 bit severity level
Low uint: 16 bit error code
```

The following is a list of errors returned by the window functions. GPI errors are listed for each call.

```
WINERR_INVALID_HWND             /* Window handle is invalid           */
WINERR_INVALID_HMQ              /* Message Queue handle is invalid    */
WINERR_INVALID_HACCEL           /* Accelerator Table handle invalid   */
WINERR_INVALID_HCURSOR          /* Cursor handle invalid              */
WINERR_INVALID_HENUM            /* Enumeration handle invalid         */
WINERR_INVALID_WINDOW_ID        /* Window ID invalid                  */
WINERR_RESOURCE_NOT_FOUND       /* Resource object could not be found */
WINERR_INVALID_SELECTOR         /* Address selector value invalid     */
WINERR_INVALID_STRING_PARM
WINERR_INVALID_HHEAP
WINERR_INVALID_HEAP_POINTER
WINERR_INVALID_HEAP_SIZE_PARM
WINERR_INVALID_HEAP_SIZE
WINERR_INVALID_HEAP_SIZE_WORD
WINERR_HEAP_OUT_OF_MEMORY
WINERR_HEAP_MAX_SIZE_REACHED
WINERR_INVALID_HATOMTBL
WINERR_INVALID_ATOM
WINERR_INVALID_ATOM_NAME
WINERR_INVALID_INTEGER_ATOM
WINERR_ATOM_NAME_NOT_FOUND
WINERR_INVALID_HWND
WINERR_INVALID_HMQ
WINERR_INVALID_PARAMETER
WINERR_WINDOW_LOCK_UNDERFLOW
WINERR_WINDOW_LOCK_OVERFLOW
WINERR_WINDOW_LOCKED
WINERR_WINDOW_NOT_LOCKED
```

Chapter 3
User Interface

3.1 User Interface Shell

The following sections describe in detail the Appearance and Function of the Presentation Manager User Interface Shell.

In general, the Presentation Manager Shell aims to present on the screen all the functions available in the system. In complete contrast to the simple MS OS/2 user interface, where very few of the objects and functions of the system are visible on the screen, the Presentation Manager Shell can show a visual representation of all objects in the MS OS/2 system and the functions which operate on them. The approach used is an 'Object-Action' one, where the user selects an object to work with and then chooses an action to perform on it. Direct manipulation techniques are used by the Shell, such as dragging objects around, and selecting actions from pop-up menus.

One aim of the Presentation Manager Shell is to reduce the need for the user to read manuals. In part this is achieved by the user's ability to see all system function on the screen. Another contribution to this aim is the consistency of use of the input devices - Keyboard and/or Mouse. The user needs to understand only a very few concepts about these devices to use the system. An important part of the Shell is the on line Help facility which it provides - again reducing the need for reference manuals.

A further important aspect of the Shell is that its user interface exemplifies the user interface which should be used by applications. In conjunction with the Presentation Manager programming interfaces (which the Shell uses), the Shell encourages the consistent end-user interface which is a prime aim of the Presentation Manager product.

Consistent with these objectives, the Shell sets out to provide the following capabilities:

1. Provide structured access to the files found on the user's system.
2. Provide a simple, intuitive approach to filing system management:
   - high and low level topology/index to the filing system
   - basic manipulation operations (copy, rename, etc.)
   - access to files stored in the system with direct manipulation from the mouse
   - visibility of filing system while running applications
3. Allow for the creation of easy-to-use systems - that is, to provide a way to configure a system to allow the naive users to focus and work with a predetermined set of applications and files.

3.1.1 General Features of the Shell

The basic functions included in the Shell are:

1. *The Screen Layout* - Windows
2. *The File Cabinet* is the index, viewer, and repository of objects related to the user's data and program storage. Contained within the File Cabinet there are:
   - Drives
   - Directories
   - Programs
   - MS OS/2 files
   - STARTUP (programs)

   The file system is hierarchical - it consists of drives, which contain a mixture of MS OS/2 files and directories. Directories in turn may also contain such a mixture, yielding a tree of arbitrary depth. By opening a drive or directory in the tree, the user reveals a window showing the objects found in that drive or directory.

   STARTUP, contained within the File Cabinet, allows the user to easily manipulate installed applications; for example, starting a program running.
3. *The Task Manager* is the window that provides access to, and control of, objects that exist in the working environment. In addition, it provides general control of the user's session, defined as that period of time that the user is interacting with the system.

   A window in the workspace can easily be brought to the front using the Task Manager, by selecting its name from the list presented in the Task Manager window. In addition, it may also be possible to perform certain other operations on the selected task, such as closing it.
4. *The Control Panel* allows users to set their workstation configuration and other system related parameters.
5. *Printer Services* provides output control for any device other than the screen.
6. *Startup Editor* allows new programs to be defined, and existing programs to be changed.

These functions are described in more detail in the following sections.

The initial implementation of the Shell includes direct manipulation in the File Cabinet using the mouse.

The data displayed on the screen is divided into a number of windows. Each window encloses the data belonging to one part of the interaction between the system and the user. The windows are rectangular and can overlap, giving the appearance of papers on a desk top. This means that one window obscures the part of another "underlying" window where they overlap.

The screen can have many windows displayed at once. A particular use made of windows is the display of menus and dialogs, where the window is displayed for a short time while the user makes a choice or inputs some data. The window is then removed to avoid cluttering the screen.

3.1.2 The Pointer

The Pointer is normally displayed only if there is a mouse attached to the system. However, it is also displayed on mouse-less systems at certain times to indicate to the user that some particular action is taking place. For example, an hour glass is shown to indicate that the user must wait while a lengthy operation is in process.

The pointer is a small shape which reflects mouse movements on the screen. The pointer is displayed 'on top' of the other data on the screen and so is always visible. The position of the pointer marks the user's center of interest and activity on the screen. Its position is used when the mouse buttons are pressed; for example, to select an item of data on the screen.

The shape of the pointer can vary as it travels over the screen. There is a system default shape - an arrow - but each window on the screen can have its own pointer shape defined. When the pointer position moves into a window which has its own pointer shape, the pointer changes to that shape. Similarly, when the pointer position leaves the window, the pointer shape changes back to the system shape or to the shape defined for another window.

3.1.3 Selection Cursor

The Selection Cursor is used to indicate a selectable item that the user can select. It is displayed whether or not a mouse is attached and is in addition to the pointer. It marks the whole of a selectable item as being the center of interest for the user. For example, it can indicate one item in a list of menu items which the user wants to select. The Selection Cursor can be moved by the mouse or by keystrokes from the keyboard.

3.1.3.1 Selecting Items

Many items on the screen are *Selectable.* This means that the user can choose the item (*Select* it) and then perform an action on this item.

Selection is performed in a standard way and is not dependent on the kind of item involved. The way of selecting an item differs between the mouse and the keyboard. Generally, a single item is selected at a time and then some function performed. However, in some cases, multiple items can be selected and the same action performed on all of the items. Multiple selection involves a different way of using both the mouse and the keyboard. All these methods are described in the next sections.

3.1.3.1.1 Single Selection

Keyboard

The user can move the selection cursor around the selectable items in the window which has the input focus. This is done using the arrow keys, which move the Selection Cursor to the next selectable item in the direction indicated by the arrow. To move from one group of controls to another, the Tab key is used.

Movement of the Selection Cursor normally involves auto-selection of the items. That is, when the Selection Cursor is moved, the item onto which the cursor moves is selected and the previous item is deselected.

Mouse

The user can move the mouse until the Pointer lies over a selectable item and then press button 1 down. The item is displayed in reverse video. When button 1 is released, the item is selected. Any other item(s) selected are deselected.

The press and release of the mouse button in this way is termed a "click".

The following action occurs for menu bars and pop-downs. The user holds down button 1 and moves the pointer around the screen. When the Pointer moves away from the original item, it stops being shown in reverse video. If the Pointer tracks across other selectable items, whichever item is under the Pointer highlights. This allows the user to browse around the selectable objects. Whichever item is under the Pointer when button 1 is released is selected. If no item is under the Pointer, no selection occurs.

This use of the mouse is termed "press and hold".

For other than menu bars and pop-downs, the mouse button must be pressed to change the selected item.

3.1.3.2 Multiple Selection

In fields with multiple selection (e.g., Check boxes), the following applies:

Keyboard
: The user can move around the selectable items in the window as for single selection. The space bar is used to toggle selection of an item, and the Enter key to submit the panel.

Mouse
: The user can move the mouse until the Pointer lies over a selectable item, and then click button 1. The item is displayed as selected, other selections are unchanged. To deselect an item, the user clicks on it again.

3.1.3.3 Extended Selection

Note that not all windows allow extended selection. Some windows restrict the user to a single selection at a time. This is typical of pull-down menus, for example. For those windows that do allow extended selection, the following methods apply.

Keyboard
: To extend the selection of items using the keyboard, the user must press the space bar to change from the auto-select mode to multiple selection mode. The user can select additional items, by pressing the space bar again.

  The mode terminates when Enter is pressed, or the dialog canceled. While the mode is active, the selection cursor is displayed independently of selected emphasis, even when they apply to the same item.

  To select contiguous items, shift+arrow keys may also be used. These do not cause a mode to be entered.

Mouse
: The user can press the space bar to switch to multiple selection mode and then click mouse button 1 with the pointer over an item to select it. Shift+button 1 may also be used; this extends selection from the selection cursor to the position clicked on.

  Double clicking the mouse performs the default action on all selected items if in extended selection mode. Otherwise, to invoke the default action on all selected items the the user must hold the shift key down while double clicking the mouse.

3.1.4 Use of Keyboard and Mouse

Keyboard and mouse can be mixed for selecting groups of objects. The only exception to this is that direct manipulation of files is not available from the keyboard.

This section describes the standardized actions and their meanings. It gives the user a guide to using the Presentation Manager system and the programs which run with it. This is done in terms of the keyboard keys and mouse actions which can be used and their meanings in terms of system functions.

3.1.4.1 Keyboard

The keyboard keys are divided into several logical sections:

Alphanumeric Keys
: which include the A-Z, 1-0 and special character keys. These are typically used only for the input of data. Corresponding characters appear on the screen when these keys are pressed and they generally have no other effects.

Function Keys
: which typically include F1-F12. These are normally used to invoke particular actions. For example, F1 has the standard meaning of 'Help' and brings help information onto the screen.

Movement keys
: include the Arrow keys (Up, Down, Left and Right cursor movement keys). The Home, PgUp, End, and PgDn keys also fall into this class. These are used to cause objects to move around the screen. The typical object that they move is the Selection Cursor when the user is interacting with a list of selectable items.

Ancillary Keys
: including the Shift, Alt, Ctrl Keys. These are used to modify the meanings of other keys. The simplest example of their use is to cause the alphabetic keys to produce uppercase characters when the Shift key is held down.

Specific meanings of some of the keys are described below. This list includes all those keys with associated functions which are essential to the use of Presentation Manager:

Alt+Esc
: Jump to next task/program (includes non-Presentation Manager programs). This makes the current active application inactive and causes the next task in the Task Manager to become active.

Ctrl+Esc
: Jump to Task Manager. Causes the Task Manager to become the active window. The Task Manager list window is brought to the top. This occurs even if the active application is not in the Presentation Manager screen group, in which case the screen group is switched back to the Presentation Manager screen group first.

Enter
: This has two meanings depending on context:

- Submit the changes.
- Take the default action on the selected item(s)

Arrow Keys
: Move Selection Cursor to next selectable item. (Selects items as it moves, with deselection of previous item(s) - for Auto-Select.)

  The Up and Down Arrow keys also work in a specialized way when used with an Action Bar and its associated Pull-down menus. The Up and Down Arrow keys cause the selection cursor to move between the rows of a multi-line Action Bar. When the cursor reaches the edge of the Action Bar, the keys cause the Pull-down menu (associated with the last indicated item) to appear.

Shift+Arrow Keys
: Extended selection - input field (Swipe and type).

Delete
: Deletes selected text to clipboard, or deletes single character to right of insertion point (cursor).

Backspace
: Deletes character to left of cursor

Ctrl+Arrow Keys
: Moves to the beginning/end of fields, or words.

Spacebar
: Toggles selection status of item for multiple selection panels and also switches into extended selection mode.

Tab
: Moves selection cursor between groups of controls

F10
: Toggle to/from (Application) menu bar (same as Alt make/break)

Alt make/break
: Toggle to/from (Application) menu bar (same as F10)

Shift+Esc
: Bring Up System Menu (or remove, if already shown).

Alt+F4
: Close window (if Close is on the System Menu).

Alt+F5
: Restore window

Alt+F7 Move window

Alt+F8 Size window

Alt+F9 Minimize window (toggle)

Alt+F10
Maximize window (toggle)

3.1.4.2 Mouse

The mouse is used in two ways. It can be moved. It has buttons that can be pressed. These actions are used in conjunction to provide a powerful tool with which the user can interact with the system, using the screen to provide rapid feedback.

Some actions cannot be done with the mouse alone. The shift key on the keyboard is used to perform certain actions. The list below shows when the keyboard shift key is used.

Button 1 Click
Select item under Pointer

Shift+Button 1 Click
Extend selection to include items between pointer and previous cursor position.

Button 1 Double Click
Select item and perform default action. If in extended selection mode, add item under pointer to selected items, and perform default action on all items.

Shift+Button 1 Double Click
Add items to Selection between pointer and previous cursor position, and perform default action on all selected items.

Button 3 Click
Jump to Task Manager

Button 3 Double Click
Jump to next task

Mouse Movement
Moves the Pointer around the screen. It is used to indicate the point of activity or interest in the data presented on the screen.

Button 2
Application defined meaning - can vary from program to program.

Press and hold button 1
Drag selected object around window or screen. Meaning is context and application dependent.

3.1.5 Functions for Controlling Windows

3.1.5.1 Appearance of Windows

All main task windows are surrounded by emphasised borders when shown at any size except maximized or minimized. Maximized windows may be shown with all borders just off the screen, although conceptually still there, but in this case they may not be moved.

Minimized windows are distinct in that they are shown in iconic form. Hence minimized windows do not have normal borders.

All windows must have a title bar except:

- Minimized windows
- Maximized windows which need the whole screen

Minimized windows may be restored by double clicking on the icon.

The title text of the window is shown next to the icon when it has the input focus.

A single mouse click on a minimized window shows the System Menu. A double mouse click (or pressing Enter when the minimized window has been selected) will open the icon up to show the program.

Non-Presentation Manager programs show up as icons in the Presentation Manager screen group. A System Menu is shown when they are selected, but a screen group switch only takes place when the window is opened for use. Thus the user can browse all tasks in the system without constant switching of screen groups. Note that if a non-Presentation Manager program is selected in the Task Manager window, then the program is shown directly, rather than bringing the icon representing that program to the front of the screen.

3.1.5.2 The Shell, Windows and Tasks

User input, such as a keystroke, mouse movement or a mouse button press is either passed directly to a program, or is intercepted by the Shell. Input to the Shell may in turn generate some other input to one or more tasks.

For example, a mouse button may be pressed when the pointer is anywhere on the screen. This can cause one of the following to happen:

- The task deals directly with the pointing.

- The input focus is switched to the task's window and it then deals with the pointing.
- The Shell deals with the pointing.

3.1.5.3 The Input Focus

The Input Focus is the place to which keyboard input is directed at any time. One window at a time has the Input Focus. The window which has the input focus is distinguished by:

- Being on top of all other windows.
- Having its window title showing selected emphasis.

Input focus can be changed either by a mouse Pointer selection in another window or by use of the "Switch Task" key, or by using the Task Manager.

3.1.5.4 Window Manipulation - the System Menu

The Shell provides a set of functions to allow the user to change the shape, size and position of screen windows. These functions are contained in the *System Menu,* which the user can access by selecting the System Menu icon (small icon on left side of the title bar) with the mouse, or pressing Shift+Esc. The System Menu contains the following functions:

- Restore
- Move
- Size
- Minimize
- Maximize
- Task Manager
- (optionally) Close

Applications can add **Close** to the System Menu if they wish to support double click on the System Menu as a fast path to **Close** an application. They must still support **Exit** on the application menu in this case. *Note:* for default VIO applications the System Menu will contain **Close**, and will also contain **Mark**, **Copy**, and **Paste**.

3.1.5.4.1 Z-ordering

In considering some of these functions, the concept of Z-ordering may be useful. It is notionally the third dimension of the screen and accounts for the order in which windows overlap each other. The topmost window visible is the highest in the Z-order; the bottommost is the lowest. In terms of pieces of paper stacked on top of each other the Z-order is the depth and order of the pile. The Z-ordering also controls the jump ordering of applications.

3.1.5.4.2 Window Maximize

An application may define a size to appear when the user selects maximize. This size cannot be larger than the screen size, although neither window title nor borders need be shown if the application needs the maximum screen area.

To achieve this the user either clicks at the Maximize icon (with button 1) on the window title bar, or selects Maximize on the System Menu for that window. While the window is maximized, the Maximize icon on the title bar is replaced with the Restore icon. Maximized windows may be returned to their original size and screen position with Restore, or sized in the normal way. (If the window is resized, the Maximize icon is returned to the title bar and the Maximize command is reenabled.

The maximize key (Alt+F10) toggles. While the application is maximized it performs **Restore**.

Applications can be run with a smaller screen area, but may be maximized at the user's request. This smaller size might typically not cover the icon "parking lot".

3.1.5.4.3 Window Minimize

In order to occupy as little screen area as possible, applications may be minimized. This will shrink the application to a predefined (iconic) bitmap.

To achieve this the user either clicks on the Minimize icon (with button 1) on the window title bar, or selects Minimize on the System Menu for that window. When the window is minimized, its appearance is defined by the application, but is normally a small bit-map giving a visual clue to the program function. When a window is minimized it is moved to the bottom of the z-ordering, and the next non-minimized window is made active. Minimized windows may be returned to their original size with Restore, or double clicking on the bit-map icon.

The minimize key (Alt+F9) toggles. It performs Restore while the application is minimized.

There are two possibilities for where a minimized window goes:

- to the place it was when last minimized. Minimized windows can be moved around the screen like other windows.
- to the icon "parking lot" if it was never minimized.

In either case, minimized windows are never overlapped. They are positioned on a notional grid on the screen, and if one position is occupied, the next position to the right, (then in row above) is used.

The position of minimized windows is not related to their position when restored.

3.1.5.4.4 The Parking Lot

The icon "parking lot" is this notional grid which overlays the screen. Each grid segment is large enough to contain exactly one icon; the grid starts at the bottom left of the screen, and goes from right to left, top to bottom. All icons will be aligned to this grid pattern.

3.1.5.4.5 Change Window Size

This is achieved from the mouse by pointing at the window border and selecting one of the four sides or one of the four corners.

- If a side is selected, that side may be moved towards or away from the opposite side. The opposite side is unchanged. The window becomes larger or smaller in one dimension only.
- If a corner is selected, the two adjacent sides may be adjusted to make the window larger or smaller in two dimensions at once.

In either case, the extent of the new window borders is indicated with an outline box which moves with the mouse. When the mouse button is released, the window occupies the position and extent indicated by the box.

From the keyboard, sizing is from the System Menu. Changing the size is then achieved by use of the arrow keys to move the corner or edge in the indicated direction.

The first up/down left/right arrow kit hit will identify the horizontal and/or vertical edge to be moved.

This can result in the window borders being moved just off the screen and thus becoming not visible.

3.1.5.4.6 Window Move

To move a window with the mouse, the "press-and-hold" technique is used. The user points anywhere in the window title bar and then drags an outline box to where the window is to be positioned.

The window is redrawn when the mouse button is released.

Windows may be moved off of the screen to the extent that their title bar remains visible.

From the keyboard, the user selects "move" from the System Menu, and then moves the outline box using the cursor keys. The same restrictions on window position apply.

3.1.5.4.7 Restore

Restore returns the window to its last (unmaximized, unminimized) position and size.

Size or move operations between maximize or minimize do not affect this position. Thus the window can easily be returned to its "normal" position using **Restore**.

The user either clicks on the **Restore** icon on the window title bar, or selects **Restore** on the System Menu for that window. For mouse users double click on the title bar is a fast path for **Restore**.

The position of a window is only "remembered" when it is maximized or minimized from some intermediate position.

3.1.6 File Cabinet—Functions for Using Directories and Files

The File Cabinet is a major feature of the system for most users. It is a program which lets users display and manipulate their file system, including any network connections. The file system is represented visually.

The File Cabinet provides a range of functions which can be performed on these items, such as opening a file (which creates an instance of the appropriate application for manipulating that file), moving a file into a directory, opening a program (which creates an instance of that program, without specifying a particular file to be worked on), copying a file, etc.

The File Cabinet and its associated windows are accessible to the user while running other applications. This allows the user to locate and browse other files at any time.

Working with files and directories in the File Cabinet would normally be through direct manipulation with the mouse. For example, to move a file from one directory to another, the user selects the file and drags it to the open destination directory.

Double clicking on an object, or selecting it and hitting Enter will open that object, which gives an object-oriented appearance to the system.

To unify the concepts of a file manager and application manager (for starting programs), a special "Startup" window has been created which contains directories and programs. Programs may be given long, descriptive names, yet are viewed and manipulated in the same way as the rest of the user's file system. Information about the user's file system is presented to the user in the form of three types of windows:

- The File Cabinet Window
- The Tree
- Directory Windows

Only one menu bar is available for the entire File Cabinet. This contains two sorts of functions:

- functions which operate globally on the entire file system
- functions which operate on the contents of the current directory

All directory windows are child windows of the File Cabinet window. In the File Cabinet, child windows are created slightly smaller than the window from which they came, and are slightly offset to the right from their parent. Thus if the Tree is sized to be half the area of the File Cabinet window, all subsequent directory windows will be slightly smaller than that size. If the File Cabinet window is minimized, all child windows

temporarily disappear. If it is maximized, windows already created are not affected. If it is reduced in size, all child windows are clipped to the size of the File Cabinet.

Child windows may be sized, moved, or maximized (but not minimized).

The title bar for these windows includes the name of the Drive/Directory (truncated if necessary).

One or more objects may be selected in the list using the normal selection mechanisms.

The differences between a Directory and Drive are relatively few, with some limitations on which actions the user may take on a Drive. For example, you cannot move a drive into a directory. Similarly, drives may appear based on an explicit action taken by the user to expand the file system - such as by connecting to a remote drive on a network. The things that can be done to the contents of Drives and Directories are the same, however, so the commands available are identical.

3.1.6.1 The File Cabinet Window

This window contains no data, but represents the maximum screen size to be taken up by filing system windows. It has an action bar which provides options for the topmost child window.

3.1.6.2 Tree Window

This window consists of a main area with a representation of the drives and directories in the system. It has no menu bar, and its size is limited by the bounds of the main File Cabinet window.

On the left of the main area, each drive in the system is represented by an icon in the shape of drive or disk, accompanied by the disk volume name.

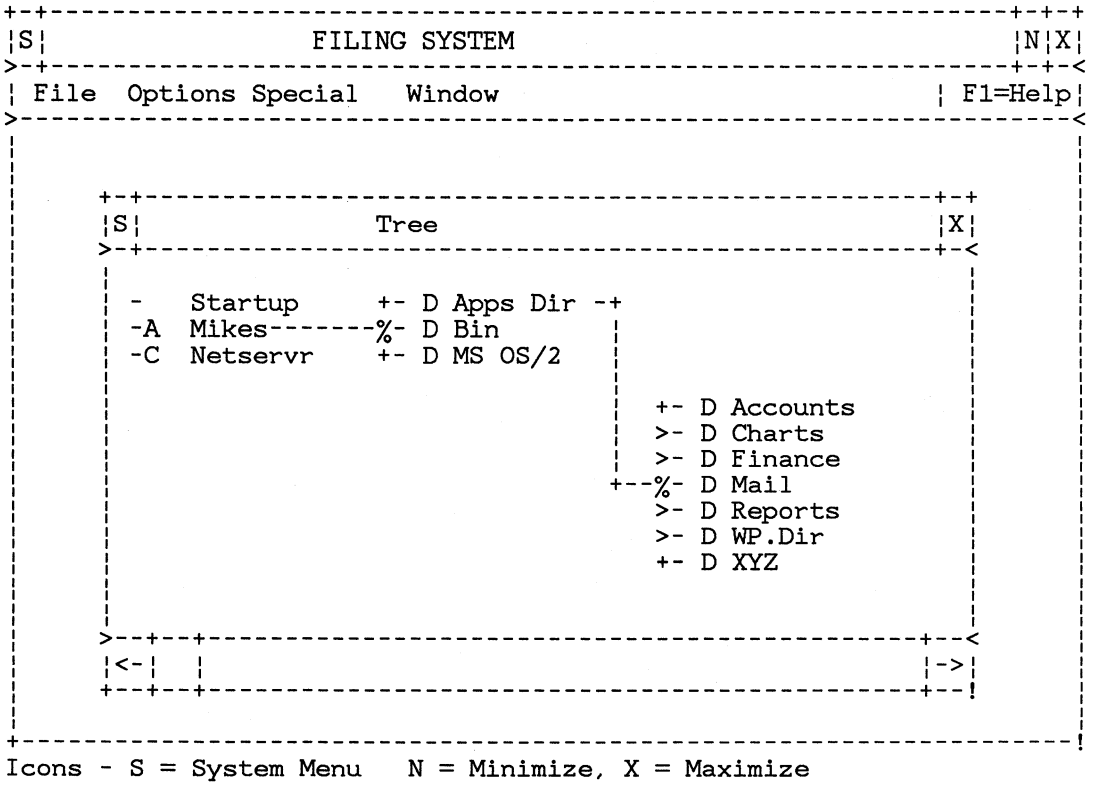


Figure 3.1 The File Cabinet with Tree

The drives are arranged in a single column in alphabetical order of drive identifier (Startup at the top). If there are too many drives to appear simultaneously, scroll buttons appear at the top and bottom of the visible portion of the column of drive icons. These allow other drives to be scrolled into view by the user.

To the right of the drive icons is a column of directory names. These are visually grouped and linked back to one of the drive icons by a series of lines or braces. The column of directories represents all the first-level directories belonging to the root directory of the disk represented by the selected drive.

If there are too many directories to fit in the window, scroll buttons appear at the top and bottom of the visible portion of the column. These allow other directories to be scrolled into view by the user.

Deeper levels of the directory tree may be displayed in the Tree window by selecting the containing directory or drive with either the mouse or keyboard. Mouse selection is done by clicking on the appropriate drive or directory. Keyboard selection is done by moving the selection with the arrow keys. The user selects the drive or directory to be displayed with up and down keys. Hitting the right arrow key causes the next level of the tree to be revealed, and the first directory in that level to be selected. Hitting the left arrow key causes the current level of the tree to be hidden. Pressing alphabetic keys causes the selection to jump to the next directory that matches the key in that level; the next level of directory is displayed, as if right arrow had been hit.

The up/down arrows always remove any tree displayed at a deeper level.

PgUp and *PgDn* scroll the current directory level by a screenful. *Home* moves to the top of the current directory level, and *End* moves to the bottom.

As with the first-level directories, subsequent levels of the tree are linked visually with their parent directory or drive by lines or braces. Each column will have its own scroll buttons as required.

The directories may be nested so deeply that all the columns of directories cannot be shown in the window simultaneously. In this case, a horizontal scroll bar appears at the bottom of the directories window. This can be used to browse the whole of the directory structure.

No files are shown in the Tree window. The files and directories belonging to a directory can be displayed in a directory window.

3.1.7 File Cabinet Functions

3.1.7.1 The File Menu

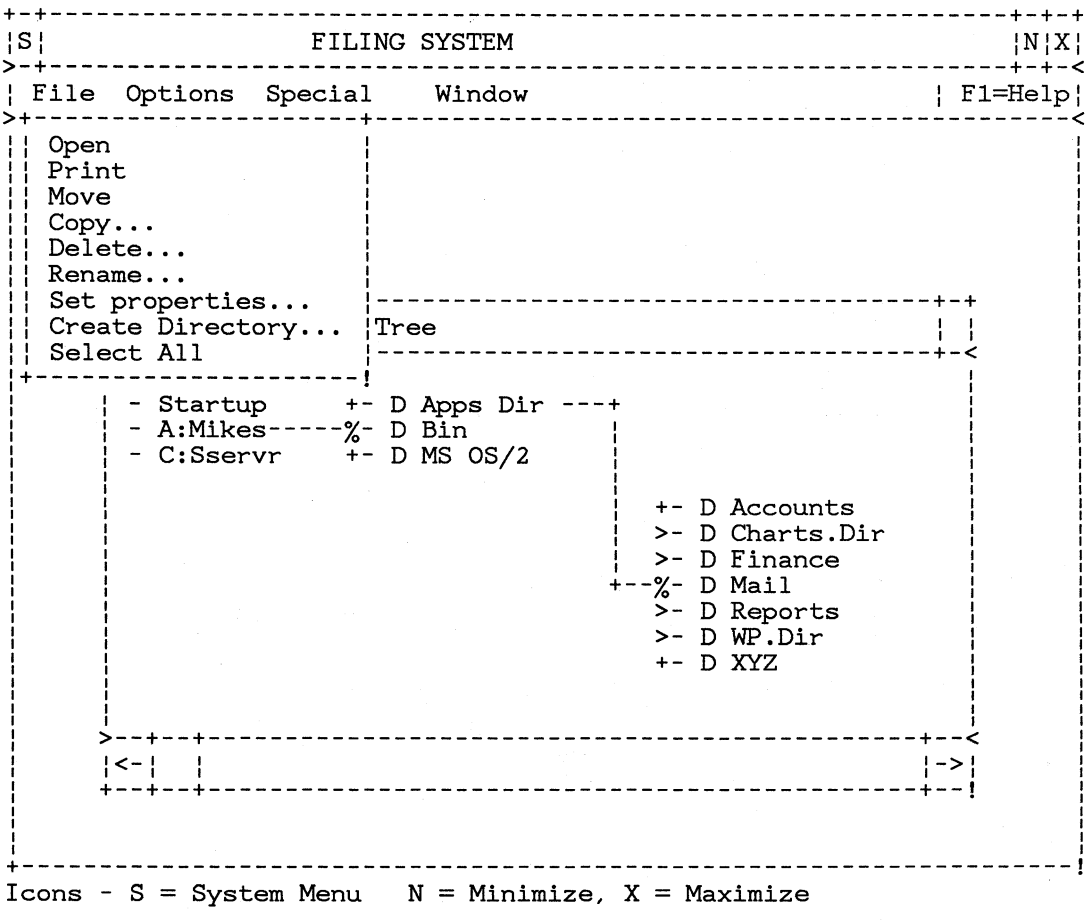


Figure 3.2 The File Pull-down Menu

The File menu includes:

Open — opens a new window containing the selected drive or directory. The new window is displayed slightly offset from the Tree window, and on top of (first in the Z-ordering) all other windows. Windows showing directory contents are created based on the size of the window from which they were created. If a window containing the drive or directory already exists, a new window is not created. Instead, the existing one is brought to the top.

The Open command is the default action for the File Cabinet window, hence double clicking an item, or hitting enter with an item selected will cause that item to be Opened.

- If the selected object is a directory, the new window is simply another directory window, containing the contents of the selected directory.
- If the object is a program, the new window contains another instance of the selected program.
- If the object is a file, and there is a default action assigned to the file then the default action is taken. (For example, .SCR might be input to a script program, when opening a .SCR file would run the script program with the particular .SCR file as an input parameter.)

  If there are multiple default actions assigned to a file type (extension), an intermediate window appears with a list of the actions. The user is allowed to select one of the actions, the window disappears, and a new window belonging to the selected action appears.

Windows other than directories are not confined to the File Cabinet, but have a size and position determined by information stored with the program being invoked. If multiple items are selected when Open is requested, the shell assumes that the intent of the user is to use all of the items simultaneously. Thus all the items are opened. Any dialog boxes are shown sequentially.

Print causes the selected objects to be printed. If there is no program assigned to the object, then the object is printed as a text file. If there is a program assigned to the file extension, the program is invoked. Programs may provide a special invocation option for printing.

Move This command is provided for users who need to move files across drives and directories. It brings up a dialog box which contains two text entry fields. The first (From:) contains the names of the objects to be moved. When the dialog box first appears, this field is filled with all selected directories and files in the File Cabinet window. The user can then type additional names. Multiple filenames are allowed in the From: field, and standard wildcard syntax is valid. The second field (To:) contains the name of the destination. It is initially blank.

The user completes the move by typing in the name of the target file or directory and hitting Enter. Leaving the field blank implies the current directory (see Create Directory for a description of the current directory).

If the user types into the From field, the typed text is added to the pre-filled text. The pre-filled text can be deleted.

It will be verified that there will be enough disk space before attempting the actual move or copy, so as not to risk running out midway through the operation.

Copy This command is identical to the Move command, except that it makes a second copy of the data.

Copy is also available by direct manipulation in a similar way to Move, but holding down the Alt key as well as the appropriate Move key(s).

Delete brings up a dialog box with a single text edit item, containing all currently selected items. The user can then type additional items. When the user confirms the delete, all items described in the text entry field are erased. Confirmation is by pressing Enter, or clicking on Delete.

Certain erase actions (drive contents, for example) cause further dialogs to confirm the operation, or ask for more details.

Rename This is used to change the name of a file, directory or file. Only a single file can be renamed.

Set Properties
allows the user to set the attributes of an object. The MS OS/2 file system attributes that may be changed are the **read-only** and/or **archive** bits, or the time and date information. Directories cannot be changed.

Create directory
This allows new directories to be created. The command brings up a dialog box to prompt the user for the name.

The directory is created in the current directory, which is the topmost window in the File Cabinet.

1. If the topmost window is the Tree, and only a drive is selected, it is the root directory of that drive. If it is showing some other directory selected, the current directory is the one shown selected.
2. If the topmost window in the File Cabinet is some other directory, then this directory represents the current directory in MS OS/2 terms.

Select all
Selects all objects in the current window. It is not valid in the Tree window.

3.1.7.2 Direct Manipulation

Using the mouse, a completely different method of moving files and directories is available. The user selects an object to be moved, holds down mouse button 1 over the object, and drags it to a previously opened directory window. When the button is released, the object is moved to this new directory.

Both the source object and part of the target directory must be visible. Objects may be dropped anywhere within the target directory with the same effect.

Both source and target directories are redrawn after the object has been moved. The details of the move are as for the keyboard version, with the same error situations, but with an additional error when the target is not a valid directory window.

Multiple objects may be moved by extending the selection in the usual way, and then holding shift while dragging. Alternatively, if the space bar has been used to switch to extended select mode, then multiple objects will be dragged without the use of Shift.

Pointer appearance during the move is that of a file, or directory, or a group of objects for extended selection.

3.1.7.2.1 Summary of Mouse Use in Direct Manipulation

The following section details the rules for keyboard and mouse interaction for direct manipulation in the File Cabinet.

The default direct manipulation operation in Presentation Manager is a move on a single object. In order to perform a non-standard move operation or standard/non-standard copy operation some interaction is required with one or more of the following keys.

ALT key
: This changes the operation from a move to a copy.

CTL key
: This is used to add an object to a non-contiguous set for a group copy/move operation.

Shift key
: This is used to add an object to a contiguous set for a group copy/move operation.

Below is a glossary of terms used in the following set of rules:

XS Extended Selection Mode. This mode is entered by hitting the *<SPACEBAR>*.

Drag Move mouse with button depressed for more than a predefined distance.

```
place=inline frame=none width=page
+----------------------+----------------------+----------------------\1c
|    Shift + Click     |    Ctl + Click       |        Click         |
>----------------------%----------------------%----------------------<
|                      |                      | Causes object to be  |
| Causes all objects   | Causes this object   | selected             |
| from prior location  | to be added to the   | mode it adds to the  |
| to current location  | set of selected      | selection, otherwise |
| inclusively to be    | objects (removed if  | anything else that   |
| selected.            | it was already       | was selected becomes |
|                      | selected).           | deselected.          |
|                      |                      |                      |
+----------------------+----------------------+----------------------!
```

Figure 3.3 Key/Mouse Click Usages for Selection and Manipulation

```
place=inline frame=none width=page
+-------------------------------------------------------------------------\1c
|                                  DRAG                                   |
>------------------+------------------+------------------+----------------<
|      Shift       |       ALT        |       CTL        | Causes         |
>----------+-------+---+----------+---+---+-------+------< object to      |
| Selects  | ALT+Shift | Causes ob-| ALT + CTL | Adds     | be select-     |
| range of >-----------< ject to be>-----------< object to | ed and in-     |
| objects  | Selects   | selected  | Adds      | set of   | vokes the      |
| from prior| range of | and invok-| object to | selected | move           |
| cursor   | objects   | es copy.  | set of    | objects  | in XS mode     |
| location | from prior| If in XS  | selected  | and      | it adds        |
| to this  | cursor    | mode it   | objects   | invokes  | object to      |
| object and| location | adds      | and       | move     | set and        |
| invokes  | to this   | object to | invokes   |          | invokes        |
| group    | object and| set and   | group     |          | group move     |
| move     | invokes   | invokes   | copy      |          |                |
|          | group     | group     |           |          |                |
|          | copy      | copy      |           |          |                |
+----------+-----------+-----------+-----------+----------+----------------!
```

Figure 3.4 Key/Mouse Drag Usages for Selection and Manipulation

3.1.7.3 Options Menu

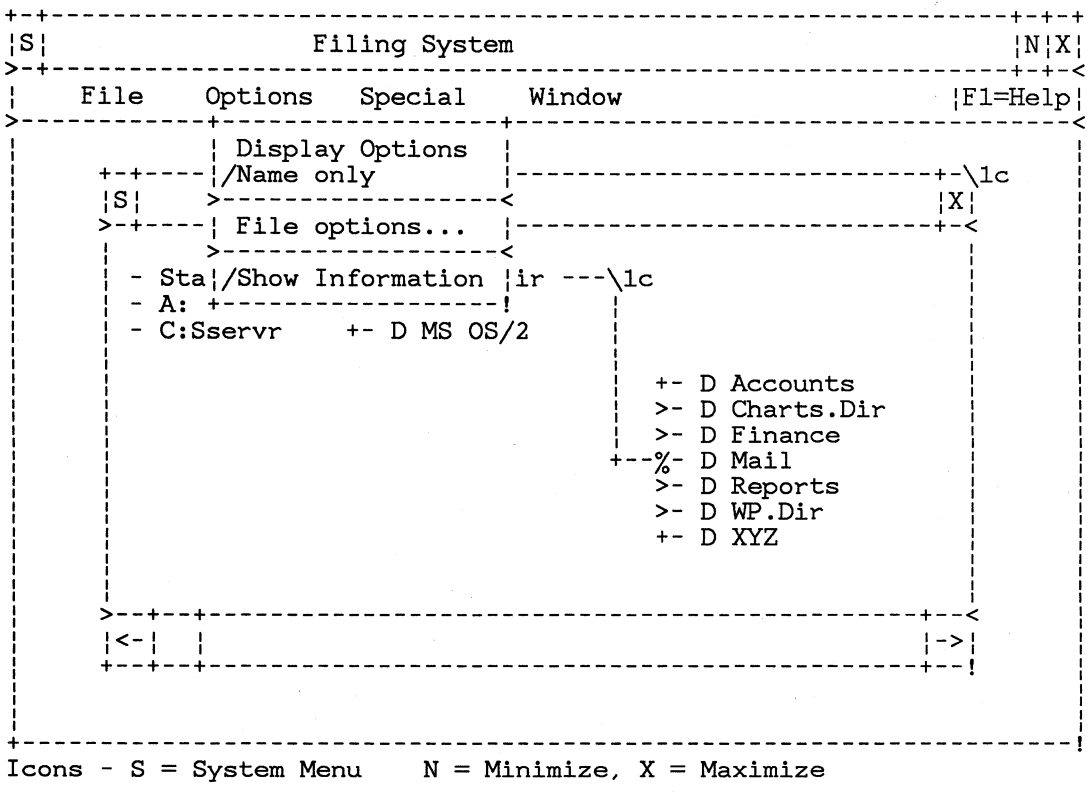


Figure 3.5 The Filing System with Options Pull-down

The options menu applies to the active child window, and (optionally) to windows subsequently created in the filing system.

Windows previously created are not affected by changing these options.

Display Options...

The Display Options dialog includes:

1. *Include*: which directories and files to display - the user can set this according to all valid attributes:
   - Name or extension - via wildcard filters
   - Type - directories, programs or files (check boxes).
   - File attributes - normal or special check boxes. Special shows a panel with hidden, read/only, system, archive check boxes. These selections show only objects with all the chosen bits set (and which qualify by the other

selection criteria).

The objects displayed are the logical intersection of each of the groups selected. The defaults are *.*, all directories, files and programs, and normal.

2. *Display order*: what information the objects should be sorted on:
   - Name
   - Extension
   - Type (directory, program, or file)
   - Size
   - Date/time
3. *What to display*: Name, date, size and attributes.

   This allows the user to set the way that the contents of directories and drives are displayed.

Name only

An extra choice is provided on the first menu as a fast path to the display of as many objects as possible. When Name Only is selected, the other options for what to display are ignored, and only the names are shown. The choice toggles.

The name is always displayed. When "Name-only" is selected, objects are displayed in multiple columns, with a horizontal scroll bar appearing if there is insufficient window space to display all objects. Otherwise, the objects are displayed in a single column, possibly with a vertical scroll bar.

File options

toggles whether certain confirmation messages are displayed.

- Verify on Copy - compare the bytes in files after a copy
- Verify on Delete - show the dialog box on all delete commands
- Replace existing file - give warning prompt
- Sub-Tree Delete - delete a directory (and everything in it) even if the directory is not empty

Show Information

toggles whether information about the drive or directory is displayed. The state is indicated by the presence of a checkmark (on) or its absence (off). If off, no information is displayed. If on, the following is displayed:

- Space used by files shown (files that have passed filter), out of total on disk.

- Number of objects visible in window OF total number of objects (files and directories).

3.1.7.4 Special Menu

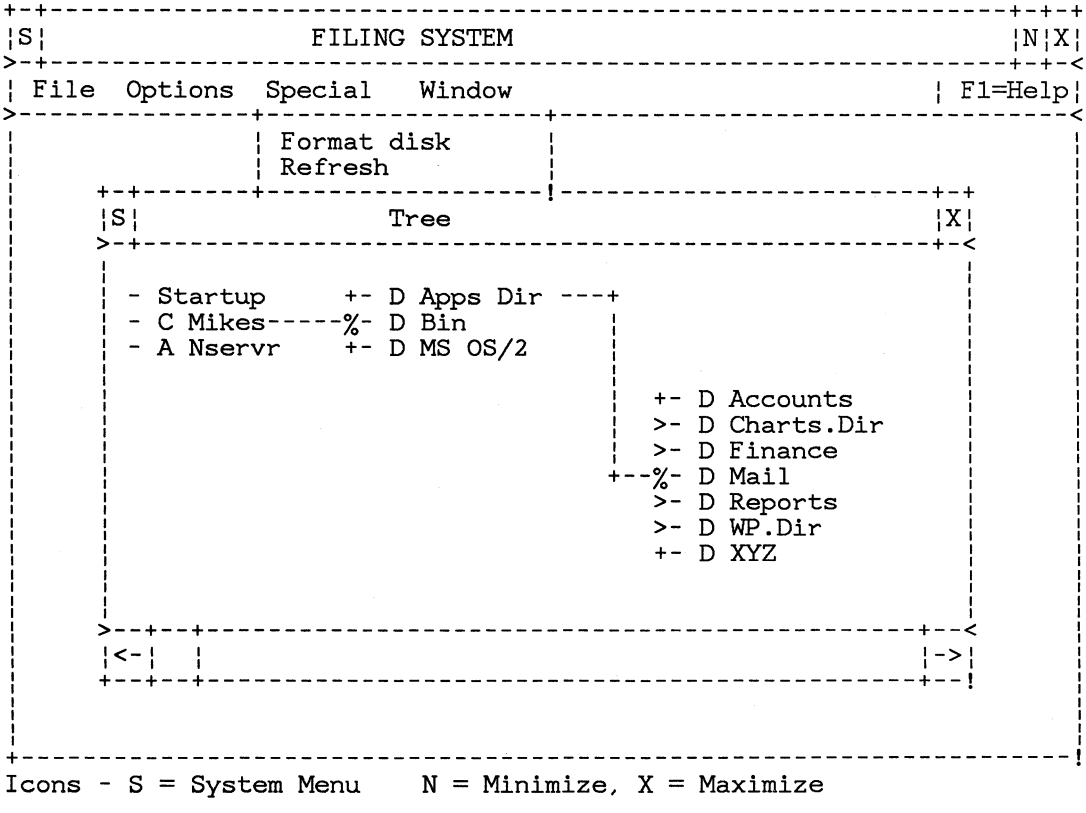


Figure 3.6 The Filing System with Special Menu

The Special menu includes commands which operate globally on the file system or disks.

Format — This allows the user to format new data disks. The command brings up a dialog box with a text field which requests the drive letter (with a logical default shown as selected, the drive name, and checkboxes which allow the user to select format options such as:

- format single/double density.

Format prompts the user for a volume label and other options in a dialog box.

Refresh — Ensures that the File Cabinet windows are all up to date.

3.1.7.5 The Window Menu

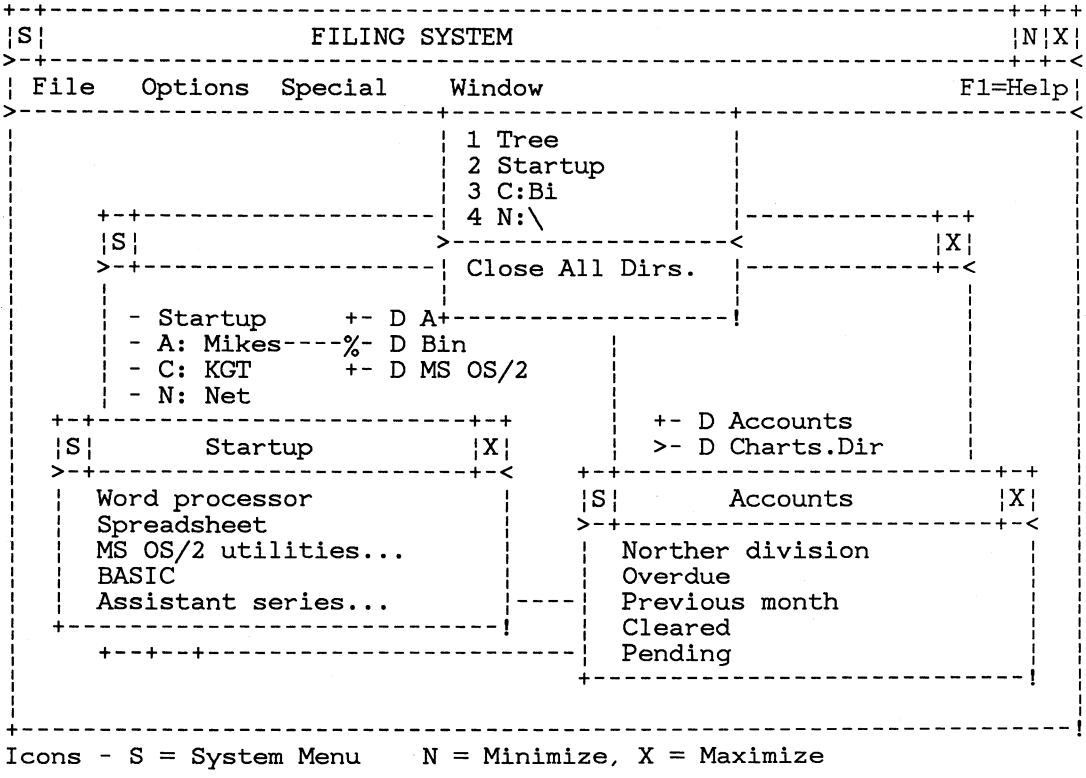


Figure 3.7 The File Cabinet with Window Pull-down

F6 The next directory in the currently displayed directories is brought to the top of the child windows and given the input focus when F6 is hit.

Directories are removed by closing them using their System Menu/icon.

1 Tree The Tree window is given the input focus and brought to the top of the child windows in the File Cabinet

Directory list 2..n
A list showing the child windows in the File Cabinet is shown in the pull-down window. If there are more than 8 directories, an option to list all the directories replaces the ninth.

The name shown is the full name of the directory displayed. For the root directory it is the drive identifier and backslash.

The names of Startup program directories are truncated to 40 characters.

Close All Directories
This commands closes all existing directory and drive windows, providing an easy way for the user to clean up the File Cabinet windows.

3.1.7.6 STARTUP Window

The Start-A-Program functionality of MS OS/2 is represented by a special STARTUP icon which appears in the Tree window along with the other Drives. Its functions and operation are essentially the same as any other directory window, with some exceptions:

- Directories are represented by program **Groups**.
- Activities must be added to STARTUP using a special application called the "STARTUP-Editor". This includes the ability to install an activity into the list, or modify an installed entry. This file is available in the File Cabinet by selecting the "STARTUP Editor" from the list of programs in STARTUP. Applications may also add themselves to STARTUP during installation.
- Entries in the STARTUP list can only be moved within STARTUP.

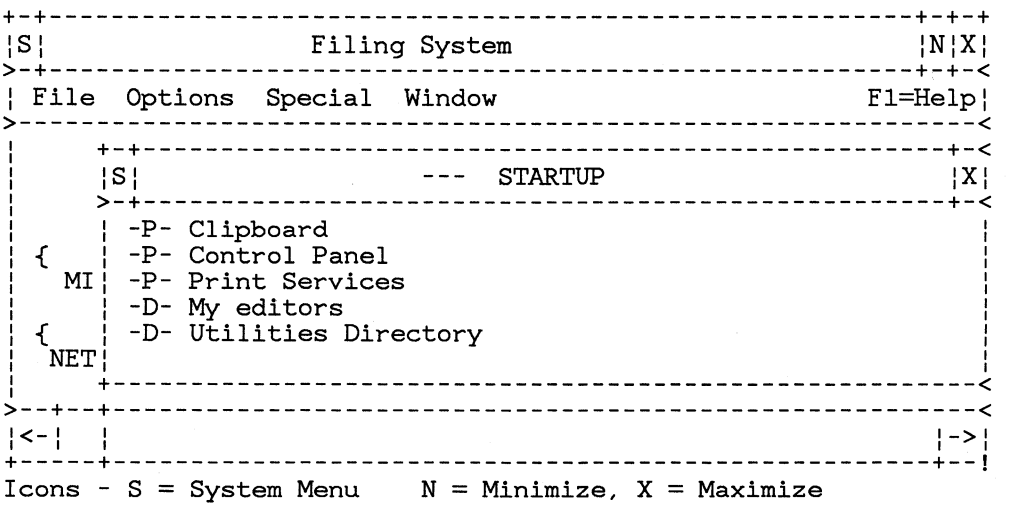


Figure 3.8 The File Cabinet with STARTUP Panel

3.1.7.6.1 STARTUP Functions

The File menu functions are:

Open — Causes selected STARTUP program to be run, or a group to be opened.

Copy — Works only within STARTUP, but is otherwise similar to normal Copy. Only one program or group may be copied at a time using this method. Names may be fully qualified using "\".

Direct manipulation works with the STARTUP programs and groups, and does allow multiple items to be moved and copied.

Move — Used only to move between groups in STARTUP. Similar to Copy.

Rename — Used to change the long name of a program.

Delete — Deletes the reference to an application from the directory. A warning is given if this is the last reference to an executable file.

Create directory — Creates a new group of applications in STARTUP.

Select all — Selects all activities in a group.

In the other pull-downs, File Options is available, plus all the Window options. All other options in pull-downs are grayed.

In the Tree, application names are displayed as the full length of the text (no trailing or leading blanks). The column widths are adjusted to support the longest entry in that group.

In windows showing the contents of program groups, the names of programs and groups are displayed in a single column. No other information is shown, apart from the directory/program icon.

3.1.8 STARTUP Editor

Most programs in Presentation Manager are added to the system using the standard installation process. The STARTUP Editor is only needed for those programs not installed in this way.

The STARTUP Editor is available from the File Cabinet, allows users to add or modify an entry in STARTUP. The STARTUP Editor consists of a main window which includes entry fields that the user enters the relevant program invocation data, including:

- Program type (Radio button choice)
- Icon file name
- Executable file name and directory
- Working Directory at invocation
- Parameters. This will have a syntax which allows prompting.
- Two-line description of the application (also provided during installation)
- Environment variables
- Program name for STARTUP
- Group in STARTUP

The last two items are prompted for by Save As...

The information supplied is stored in the PRESSERV.INI file.

All of the information which can be edited here is given initial values automatically during installation.

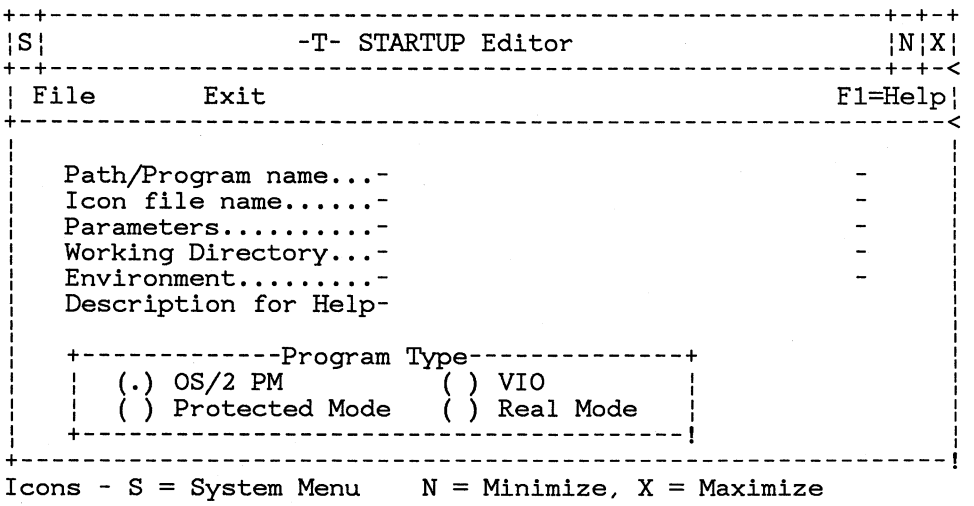


Figure 3.9 Startup Editor - Main Panel

3.1.8.1 The File Menu Option

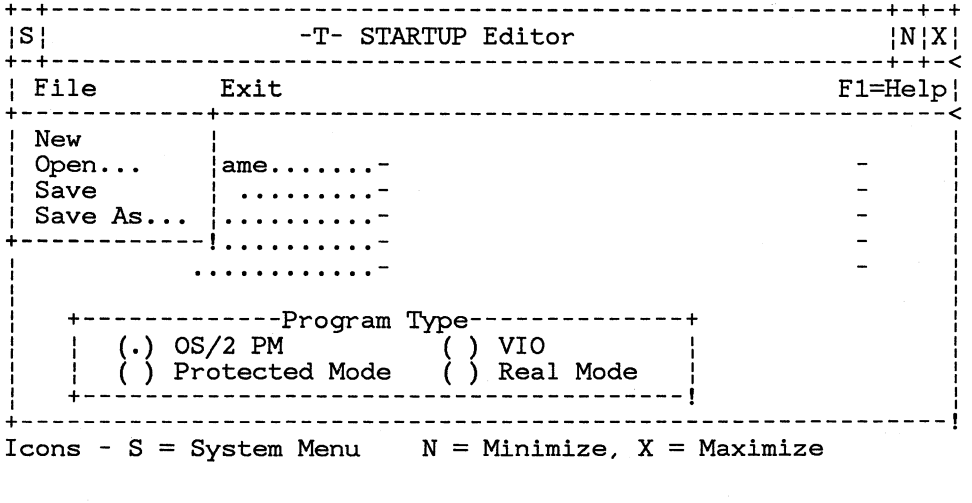


Figure 3.10 Startup Editor - File Pull-down

The File pulldown contains:

New — This command clears the any entries in the STARTUP Editor window and resets any default fields.

Open — This command is used to load a previously defined STARTUP entry. A dialog is displayed which allows the user to selected the entry from a list box.

Save — This command is used to save a STARTUP entry. If a name has not been supplied a dialog is displayed which prompts for a name, otherwise the information is simply saved.

The group name is prompted for in a similar way.

Save As — This command is used to save a STARTUP entry with a given name. A dialog is displayed with an entry field which prompts for the name. If the program already has a name, it is proposed as the default.

The group name is prompted for in a similar way.

3.1.8.2 The Exit Menu Option

This immediately quits the STARTUP Editor. If any changes have been made to the entry's information since the last save, a message will be displayed informing the user that there are unsaved changes and requesting whether to save changes. If the user responds Yes (the default), then the Save As dialog box is displayed, if not, then the STARTUP Editor ends.

3.1.9 Task Manager

Presentation Manager is capable of running many tasks at the same time. These programs can all potentially be using the screen at the same time. The user needs to be able to identify which tasks are running and to control which tasks are visible on the screen at a given time. The user also needs to close the system down at the end of work.

3.1.9.1 How to Access the Task Manager

The Task Manager is a window, which appears in the Presentation Manager screen group, that contains a list of the user's current activities. The user brings the Task Manager window into view by hitting button 3 on the mouse, hitting Ctrl+Esc on the keyboard, or by selecting the Task Manager command from the System Menu of the current application. If the Task Manager is called up while the user is in a non-Presentation Manager screen group, an implicit screen group switch is performed, and the user sees the Task Manager in an otherwise empty workspace.

The Task Manager normally contains a representation of every independent task running in the system. The representation is in text form, where the text is provided by the object, and typically matches the text displayed in the caption of the object (file name + data name).

The default Task Manager window size is large enough to display approximately 10 objects. A vertical scroll bar is displayed and can be used to move the list of objects, allowing every object in the Task Manager to be viewed, even if not all objects fit within the window. The object last worked on is selected and the list is scrolled to show the selected item at the middle of the window (unless the first or last item is visible in the window).

The appearance of the Task Manager window with several tasks running might be as follows:

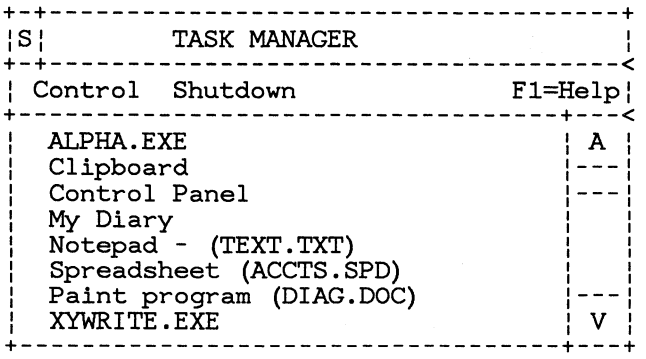


Figure 3.11 The Task Manager Window

The objects in the Task Manager are ordered alphabetically.

The entries in the Object List are selectable.

The System Menu commands include:

- Switch to
- Close (Same as Exit being selected in application)
- Terminate

3.1.9.2 Jump Ordering

The jump order round applications is their Z-order on the screen. Applications can optionally not participate in the jump sequence, but normally all applications will participate.

Non-Presentation Manager applications appear as icons in the Presentation Manager desktop, and so have a logical entry in the jump order sequence. Using the keyboard or mouse to jump to the next application will make the icon representing the non-Presentation Manager program active. To see the program itself the icon will have to be **Opened**. Note that if the corresponding entry in the Task Manager window is selected that the icon is automatically opened.

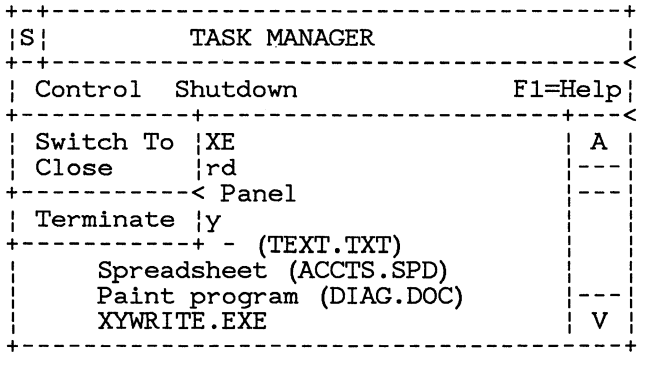


Figure 3.12 The Task Manager Window with Control Pull-down

3.1.9.3 How to Work with a Task

The user can choose to work with a particular task by:

1. Selecting the Name of the task in the list.
2. Select the Switch To command on the Control menu.

The fast way of getting a task to be the Active one is to do a doubleclick on it with button 1 or select it and press Enter.

The selected task becomes the Interactive Program.

Certain tasks can cause the Switch To selection to be grayed.

Alphanumeric keys can be used to move the selection to an object on the list when the first letter of the name matches the key pressed. If there is more than one match the selection moves to the first. If the same key is pressed again, the selection moves to the next and so on, recycling at the end of the matching section to the top. If no match is found, the machine beeps.

For *Presentation Manager* tasks, this causes the main window of the object to appear on top of the other windows in the Presentation Manager screen group. The application may choose to bring its other windows to the front at the same time. Keyboard input is directed to one of the windows belonging to the task.

For *non-Presentation Manager* programs, the Presentation Manager Screen Group is removed from the display and is replaced by the Screen Group containing the program. This occupies the whole screen and no other programs can be seen. Both Mouse and Keyboard input are directed to the program.

Non-Presentation Manager programs also show up as icons in the Presentation Manager screen group.

3.1.9.4 How to Close a Task

Some programs will not have a Close (Exit) command. The user may use the Close command to close these objects. This commands requests that the program close down normally (save data, clean up).

3.1.9.5 How to Terminate a Task

Most programs have their own methods of being brought to an end normally, through menu options or commands. Generally the user should use these methods to terminate a running program. This is advisable because the program probably needs to tidy things up before it finishes - save work away in files, for example.

However, it can happen that a program gets stuck or begins to behave in an unusual way. To allow the user to stop such a program, the *Terminate* function can also be used quit a program. When invoked the Terminate function causes a destructive shutdown, i.e, the program does not get a chance to save data or otherwise clean up.

To terminate a program in this way, select the program's entry in the Task List and then select the *Control* option in the Task Manager menu bar. Select the *Terminate* option on the pull-down menu. A warning panel is displayed. This allows the user a second chance to think about the destructiveness of the Terminate function and prevents inadvertent program stopping.

The filing system window cannot be closed or terminated.

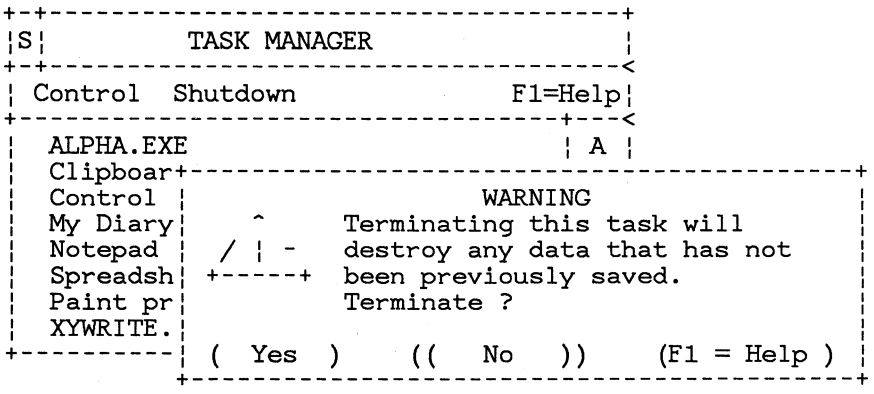


Figure 3.13 Task Manager - Terminating a Task

Either the Yes or No buttons must be selected. The default button is No (for safety), in case the user presses the Enter key as the first action after this panel appears.

Selecting the *No* option quits the Stop operation and leaves the program running.

Selecting the *Yes* option causes the program to be stopped.

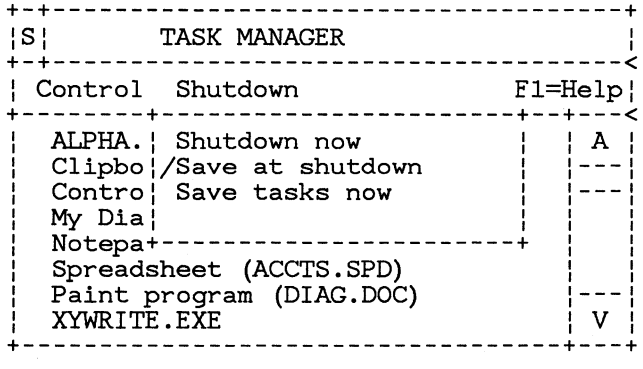


Figure 3.14 Task Manager with Shutdown Pull-down

The Shutdown menu includes:

Shutdown now
: This is the normal way to close the system down at the end of the work session. There are two variations, controlled by the next option in the menu.

  1. **Save at shutdown** *on* causes Shutdown-now to save the entire task list in terms of the applications running, and their position on the screen. Each application is responsible for saving its data and current state.
  2. Shutdown with **Save at shutdown** *off* causes all applications to end normally but no record is kept and Restarting the system will not restart the current set of applications. Before an application ends, it is expected to prompt the user to save any unsaved changes, and then to shut down.

Save at shutdown
: This toggle indicates the current action to be performed at shutdown. It is initially set off (no save).

Save tasks now
: This provides the user with an easy way to save the layout and data ready for the next IPL. This includes notifying applications so that they can be restored at least working on the same file at the next IPL. Screen window layout and current options must also be preserved.

  No shutdown is performed.

The Task Manager window is removed from the screen only following a Switch To operation.

3.1.10 Control Panel

There are many options the user has for how the system works. Most have to do with the hardware configuration, while others have to do with the Presentation Manager system's appearance. Control Panel allows the user to change these settings:

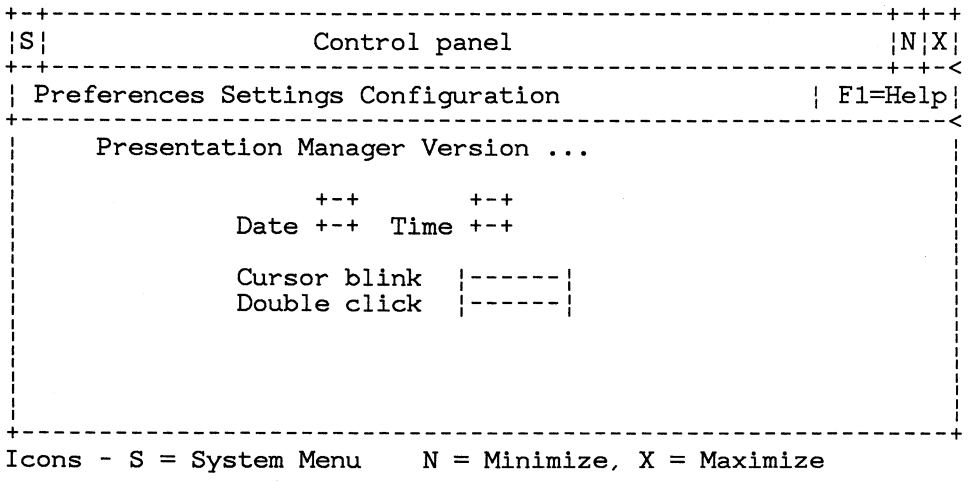


Figure 3.15 Control Panel

3.1.10.1 Main Panel

Time & Date
: The user can set the time and date (Entry fields). This will set the internal hardware clock.

Double click
: Individual mouse clicks must be received in order to generate a doubleclick.

Cursor Blink
: This changes the rate at which the pointer flashes. Cursors which flash too quickly tend to be distracting, while if they are too slow, then they are hard to spot.

3.1.10.2 Preferences Pull-down

Sound On or Off
: Allows the user to turn off sound (Check box)

Screen Colors
: The user can select which colors are used in various parts of the system. For example, the items below can be changed. This list is not considered to be exhaustive.

- Window Background
- Window Text
- Scroll Bar
- Scroll Arrows
- Scroll Elevator
- Active Title Bar
- Inactive Title Bar
- Title Bar Text
- Window Borders
- Menu Bar
- Menu Text
- Screen Background

Border sizes
: The user is able to select border width. Novice users may want wider borders to make it easier to manipulate the borders directly. More experienced users may want to shrink the borders.

Logo on/off
: The user can suppress all logo displays. Default is no suppression. (Check box).

Mouse Buttons
: Right handed people tend to want their mouse buttons from left to right (1-2-3), whereas left handed people want the opposite (3-2-1). Control Panel allows the user to select which he or she wants.

3.1.10.3 Settings Pull-down

Printer drivers
: Options to select additional printer drivers (Entry field)

Printer defaults
: Options to select default printer and settings (List boxes)

Print spooler
: Options to select spooler (Entry field)

Communications
: Options to select Baud rates, etc. (Entry fields, radio buttons)

Ports
: Options change Comm1, etc. (List boxes)

3.1.10.4 Configuration Pull-down

Global data:
In the Presentation Manager system, "PRESSERV.INI" is not a text file, and if the user's system configuration changes, it is not possible for the user to edit the changes into "PRESSERV.INI" with a text editor. Therefore, the Control Panel must account for all such possible hardware changes. This includes changes to CONFIG.SYS. This includes changing the type ahead buffer size, and the autorepeat rate of the keyboard. The more commonly changed options are shown individually with entry fields, others are shown in list boxes.

Path: The user can set the initial path for the system.

Fonts: It is possible to purchase new fonts for Presentation Manager and to tell Presentation Manager that they exist.

International Settings:
The user can change the time/date format, the currency symbol, and other international characters.

Default action definitions:
The default Open action and Print action may be specified according to file extension. The panel allows this for each file extension the user wishes to define, plus a default to be used by others, or the option to disallow certain file extensions.

When applications are installed, they may wish to prompt the user as to whether they should appear for certain file extensions.

List boxes and entry fields allow this.

Miscellaneous system variables:
Users can change any system variable through the control panel. Applications can define variables in the initialization file themselves, and list boxes and entry fields will be available to change these.

3.1.11 Cut, Copy and Paste

Presentation Manager provides copying and moving using the cut, copy and paste metaphor. This allows an object/action approach to be applied to copying, rather than the action/object approach of "copy what, to where".

To the end user, the appearance of cut and paste is as follows:

1. Mark the object to be copied or moved by the normal selection process.
2. Choose either cut (to remove it from the file to the clipboard), or copy (to copy it). In entry fields, cut may be performed using the Delete key, and Copy using Ctrl+Insert.
3. In the target file (which may also be the original file), select the target position.
4. Select paste. In entry fields, Paste may be performed using Shift+Insert. This replaces selected text with the clipboard contents.

Typically, these functions are provided on an Edit menu in each application (as in the STARTUP Editor), and the user can use them both inside and between applications.

3.1.11.1 Clipboard Mechanics

To create this easy-to-use environment, the clipboard is set up to accept any number of different data formats, several of which it can hold simultaneously.

When a new application wants to copy the data, it looks at the data formats available, and chooses one that it understands.

The data itself may be in the clipboard already, or it may be sent as the result of a message from the target application to the source one.

Applications should use standard data stream definitions, since these will be supported by printers, plotters and editors.

3.1.11.2 Copy and Paste for VIO Applications

A simple form of cut and paste is provided for default VIO applications.

Copy — Areas of screen can be MARKed with an option available from the System Menu, and the contents of the marked area copied to the clipboard in text format.

Paste — will replay text as if it were being keyed into the application.

3.1.12 Initialization

3.1.12.1 The Initial View of the System

The view of the system when it is IPLed for the first time is that of the File Cabinet with the root directory of the default drive displayed in the Tree.

Also displayed is an open directory showing the root level of STARTUP.

This view will remain the initial view until it is replaced by the user saving the current tasks.

3.1.12.2 The Initialization File

The initialization file, called PRESSERV.INI, is a Binary file, i.e., it is not human readable and cannot be changed easily using text editors. The file is hidden, since it is not intended for direct use by end-users. Its use is mainly by the Presentation Manager system, which does provide some API functions for reading and changing its contents.

The initialization file contains all non-volatile task information. This includes files installed in STARTUP, open directories and running files, and system defaults for both MS OS/2 and Presentation Manager.

The system will boot without the initialization file, but default system settings for colors, display device, etc., will be used.

If errors are detected within PRESSERV.INI, the system will attempt to function correctly, but some strange behavior may be noticed. For example, applications that the user knows have been installed not known to STARTUP. The user will be informed of the errors, whereupon it might be necessary to restore the system from a backup copy.

On network systems there may be a local copy of PRESSERV.INI but a global copy of Presentation Manager.

3.1.13 Help Facility for the Shell

The purpose of Help is to provide information to the user which aids in the operation of the shell. When the user requests Help, information regarding the item selected in the current context is displayed. The user can also request an index of available Help topics, request General Help, or request information on the functions assigned to keys.

3.1.13.1 Invoking Help

The user can request Help by either pressing the F1 key, or by clicking the mouse pointer while it is on the F1=Help choice on the Action Bar. After doing one of these actions, a secondary window will be displayed, and it will contain a panel of information which pertains to the item on which the selection cursor is currently positioned.

Note that more than one item may be selected on the panel, but the initial help will relate to the item where the selection cursor is situated (generally the last item selected).

3.1.13.2 The Help Window

The Help window is movable and sizable, and will be the topmost window when it is active. It contains four pushbuttons along its bottom, and these make up the Common Actions Area. There is also a vertical scroll bar along the right side of the window, and this can be used to scroll the Help panel which is displayed in the window if it is too big to fit. If the window is sized smaller than its default size in the horizontal direction then the text is clipped, there is no horizontal scroll bar. The window must be resized to display the full text.

In the window's title bar appears the application name. Here is a picture of a typical Help window:

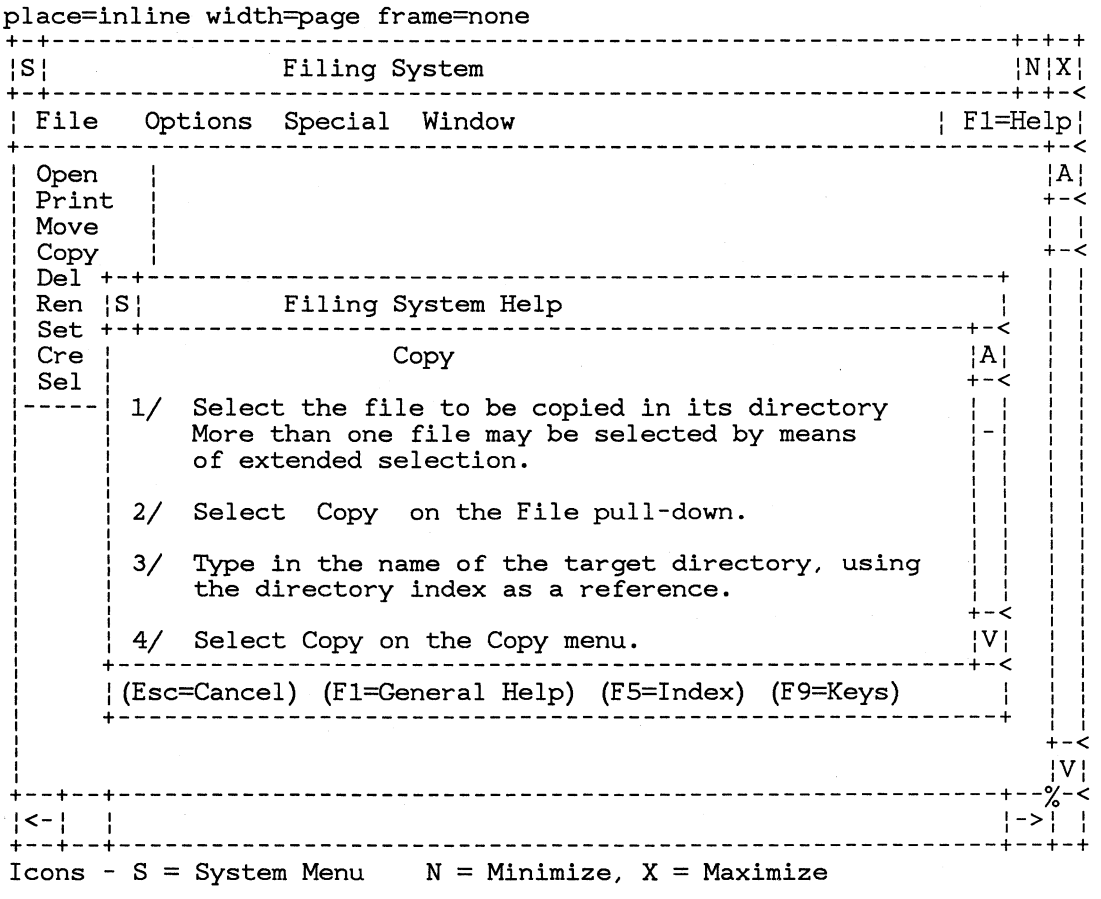


Figure 3.16 A Sample Help Window

3.1.13.3 Help Interactions

When a help panel is displayed in the Help window, the user can either use the mouse to scroll the text with the scroll bar, or can use the arrow keys, Home, End, PgUp, PgDn, Ctrl+Home, and Ctrl+End keys to scroll the text.

The pushbuttons at the bottom of the window, in the Common Actions Area, perform the functions described below. The user can perform the actions by either pressing the pushbutton with the mouse pointer, or typing the key whose label is on the desired button, i.e., F1, F5, F9, Esc.

3.1.13.3.1 F1=General Help

Displays a general Help panel describing what the panel is for and the concepts behind it. This is not help on the Help facility.

3.1.13.3.2 F5=Index

Displays a selection panel (or listbox for simple applications) which lists all of the available Help topics. The user can select a topic from the list, and it will be displayed. When the listbox is displayed, the F5=Index pushbutton changes to read Enter. The user can select a topic by either pressing the Enter key, pushing the Enter button, or doubleclicking the mouse on a topic in the listbox. When the listbox is removed, the Enter pushbutton is changed back to read F5=Index.

3.1.13.3.3 Shell Help Index

For each of the Presentation Manager Shell "Applications"

- File System
- Task Manager
- Control Panel
- StartUp Editor
- Printing Services

the HELP index will contain the following -

- A list of all the available HELP topics for the application
- An item to select help on the System Menu
- An item to select help on general controls (e.g., Scroll bars)
- An item that states that general help is available from the File Cabinet Help Index

In the File Cabinet as well as the above items there will be a list of Global Topics, e.g., Setting screen colors, Printing files.... which will contain a brief description of how these things can be done and say which application needs to be started.

3.1.13.3.4 F9=Keys

Displays a Help panel describing the functionality of all the keys available to the application (without Help displayed).

If the user asks for help on keys while in an application, the help will display the key functions for that application. The user will be advised that the application must be the interactive window before the keys will work as defined in the panel.

3.1.13.3.5 Esc=Cancel

Removes the currently displayed help window.

3.1.13.4 Additional Notes about Help

If there is no selection cursor when Help is requested, a general Help panel providing information about the currently active window will be displayed.

When the Help window is displayed, it becomes the interactive window.

During pull-down interaction with the mouse, Help must be selected using F1.

The Help panel exists until it is removed by the user, or its containing application is closed.

If the application is minimized while Help is being displayed, the Help panel is removed. If the Help panel is required when the application is restored, F1 must be pressed again.

Help is available by hitting F1 while a Window is minimized and has the input focus. Help for the System Menu will be displayed which will contain help on how to restore the window.

Help panels are not constrained within the application main window.

Examples will be included on Help panels where appropriate.

3.1.13.5 Help on Items in STARTUP

A two-line description is supplied with applications, and automatically included at installation. It may be edited using the STARTUP Editor. This brief application help will be displayed if **Help** is requested in the File Cabinet while an application is the selected item.

No index is provided for this Help.

Chapter 15
Toolkit Utilities

15.1 Dialog Box Editor User Specification

This document gives a user specification of the Dialog Box Editor, an MS OS/2 Presentation Manager application. It describes the physical appearance of the application when running under MS OS/2 Presentation Manager, how files are used with the application, and how the user interacts with the application, i.e., what the assorted commands do, and how to edit dialog boxes.

The MS OS/2 Presentation Manager Dialog Box Editor lets you design dialog boxes on the display screen and save a definition of the box in a resource file. The definition of the dialog box can be included with other resource definitions in your application's resource script file. When you create a dialog box, you create the box outline, put controls and text for the controls in it, and define the way the user will access the controls.

15.1.1 Using Files with the Dialog Box Editor

This section describes the files produced and used by the Dialog Box Editor and how to use these files with other programs such as the resource compiler, your compiler, and your linker. The actions bars and strings that make up the user interface for an MS OS/2 Presentation Manager application generally are produced from a resource definition file, a text file which has the extension .rc. The application's dialog boxes are defined in a text file that has the extension .dlg and are included in the resource definition file with the rcinclude directive. These files are processed by the MS OS/2 Presentation Manager Resource Compiler rc. Rc produces a binary resource file with a .res extension and also is used to attach the resource file to the application's executable .exe file.

The Dialog Box Editor reads two types of files and produces three types of files. It reads the application's .res file, modifying the Dialog Box Resources in it, and writes out the modified .res file. When the Dialog Box Editor writes out a .res file, it also produces a .dlg file giving the text resource definition of the dialog boxes in the .res file. The Dialog Box Editor will use the symbolic equivalents rather than the numbers where such constants are correctly contextually defined. Finally, it can also read in and write out an include file with the .h extension which is used to define symbols which can be used in place of numbers. These symbols are defined with the #define C preprocessor directive.

When the Dialog Box Editor writes a .res file, the file contains the name of the include file that was used with the .res file and all changes the user made to the Dialog Box Resources but leaves all other resources unmodified. If the .rc file is subsequently modified, it will have to be compiled with rc. If the Dialog Box Editor didn't save the resource definition text for the modified dialog boxes, they would be lost upon recompiling with rc. That is what the .dlg file is for. If you keep all your dialog box

definitions in a .dlg file with the same name as your .rc and .res files and rcinclude the .dlg file in your .rc file, the resulting .res file will always be up to date, whether it was last produced by the Resource Compiler or by the Dialog Box Editor. Note that the Dialog Box Editor never reads the .dlg file, it only writes it, hence comments in the .dlg file cannot be preserved.

The .h file produced by the Dialog Box Editor allows you to refer to controls by symbolic names rather than numbers. Each control in each dialog box has an ID Value associated with it. By using the Include File feature of the Dialog Box Editor, you may associate an ID Symbol with each ID Value. This symbol will then be defined in the .h file by:

```
#define IDSymbol          IDValue
```

The inclusion of this .h file in your .rc and in your C source files using the #include directive then allows you to refer to controls by their ID Symbols rather than their ID Values.

There are a few caveats when using the .h include files. First, the Dialog Box Editor only reads and writes symbolic constant declarations. Thus, if you have anything else in the file, such as comments, structure definitions, macros, or variable declarations, they will be lost. So, it is best to have a separate file specifically for the symbolic constants used in dialog boxes. Next, although it is possible to have more than one ID Symbol for a given ID Value, the results may be confusing because only the number is saved in the dialog box resource and the Dialog Box Editor has no way of knowing which symbol to use for that number. You will be warned if you attempt to assign more than one symbol to a given number, at that time, choose CANCEL and try another ID Symbol. Finally, if you want to use the ENTER and ESCAPE keys in the standard ways, you should only use the ID Value 1 for buttons associated with the ENTER key and ID Value 2 for buttons associated with the ESCAPE key. The reason is that whenever the ENTER key is pressed, MS OS/2 Presentation Manager automatically sends a WM_COMMAND message with ID Value 1 when the ESCAPE key is pressed, MS OS/2 Presentation Manager sends a WM_COMMAND message with an ID Value of 2. The default ID Symbols for the ID Values (in wincalls.h) are DID_OK for 1 and DID_CANCEL for 2.

15.1.1.1 Codepage Support

The Dialog Box Editor will create resouces in a specific codepage, and flag the codepage in the resource. This codepage will be the codepage in which the Dialog Box Editor application is started.

A warning message will be issued if the Dialog Box Editor detects that an existing resource it is editing is flagged in a codepage other than the codepage in which the editor is running. However, after this warning, any new resource written will be flagged with the current codepage in which the editor is running.

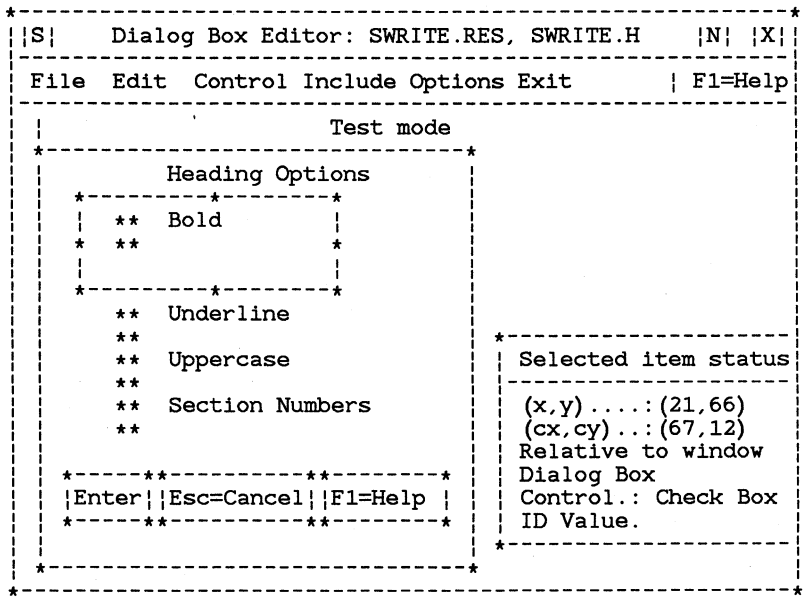
15.1.2 Application Appearance

15.1.2.1 Main Window

The main window consists of the following parts:

1. Editing area
2. Selected Item Status window
3. Panel Title

Note: The following picture gives a good representation of how the application will look. The exact appearance will depend on the final appearance of MS OS/2 Presentation Manager.



S is the system icon
X is the maximize icon
N is the minimize icon

Figure 15.1 Presentation Manager Dialog Box Editor

15.1.2.1.1 Editing Area

The editing area is where the dialog box will be created and modified. It is the client area of the application's window. The dialog box being edited may extend out of or be completely out of the editing area.

15.1.2.1.2 Selected Item Status Window

When you start the Dialog Box Editor, you will notice a small window labeled "Selected Item Status" in the lower-right corner of the screen. The Selected Item Status window stays on your screen as you edit a dialog box and supplies you with information about the dialog box and the controls in it. When you make a change to the dialog box or controls, the change is reflected in the Selected Item Status window. If necessary, the Selected Item Status window can be moved out of the way of a dialog box you are working on.

The Selected Item Status window displays the information shown in the following list. All measurements in the Dialog Box Editor are given in dialog units. For horizontal distances, one dialog unit is equal to 1/4 the width of a character in the system font (a fixed pitch font). For vertical distances, one dialog unit is 1/8 the height of a character in the system font. By restricting measurements to dialog units, it is possible to make dialog boxes appear the same on different display devices, relative to the text in the box.

(x, y) Displays the position on the lower-left corner of the dialog box or control you have selected.

(cx, cy) Displays the width and height of the dialog box or control you have selected.

Relative
Shows how the selected item is positioned. If the selected item is a control, this will always be "to Window". If the selected item is the dialog box, then it can be "to Window", "to Screen", or "to Mouse".

Control Shows the type of control you have selected (for example, Radio Button or Check Box). If the dialog box was selected, this part of the Selected Item Status window will read "Dialog Box".

ID Value
If a control is selected, the ID Value or ID Symbol is displayed. If the dialog box is selected, its name is displayed.

15.1.2.1.3 Panel Title

The panel title will be "Mode:" followed by the mode the Dialog Box Editor is currently in. There are two edit modes and a Test mode. Test allows testing of the controls in a dialog box. The two edit modes are Work (which allows full editing) and Translate (which allows limited editing). Two possible sub-modes of Work are Group (denoted by "Work/Group" and which allows moving groups of controls) and Copy (denoted by "Work/Copy" which allows duplicating individual controls). The Group and Copy sub-modes are mutually exclusive. The Group sub-mode is also available for Translate.

15.1.2.1.4 Instructions

There will be no instructions in the Dialog Box Editor's primary window.

15.1.2.2 Title Bar

The Dialog Box Editor's primary window title bar will contain the application name, "Dialog Box Editor:" followed by the names of the resource and include files being edited.

15.1.2.3 Logo Panel

When the application first starts up, it will read WIN.INI and then either

1. display the logo panel and wait for the user to respond,
2. display the logo panel for the specified period of time, or
3. go directly to the application according to the user's wishes as specified in WIN.INI.

If the logo panel is displayed until the user responds, the following logo panel will be displayed:

```
Logo Panel
```

If the logo panel will be displayed for a specific period of time, then the last line of the panel:

```
Press Enter to continue or Esc to quit.
```

will not appear on the panel.

15.1.3 Main Window Interaction

All the creating/editing of the dialog box is done in the main window with the use of the mouse. Actions Bar choices may be accessed either with the mouse or keyboard.

15.1.3.1 Resizing the Main Window

When the window is resized, the Selected Item Status window is moved back to the bottom right-hand corner of the window. Sometimes, the dialog box being edited will extend beyond the application window, especially if the application window is resized. This is because the position of the dialog box, either relative to the application window or relative to the screen, is saved as the position of the dialog box in the .res file.

If the dialog box is positioned relative to the mouse, then its position in the Dialog Box Editor will be maintained.

15.1.3.2 Dialog Box Manipulations

Once a dialog box border is up on the screen, it can be moved, expanded, or shrunk. To perform any of these actions, the dialog box borders must be selected. This can be done by clicking the mouse on a blank area inside the dialog box; the mouse pointer will be a white arrow in the areas which will select the dialog box. When the dialog box is selected, eight handles (small black rectangles) will appear on the boundaries, as shown here:

```
Outline of a Dialog Box
```

15.1.3.2.1 Moving the Dialog Box

To move the dialog box, select the dialog box and then press mouse button one inside the dialog box; the pointer will be a plus sign (+) in the valid areas for this. While holding the mouse button down, drag the mouse and a skeleton of the borders will appear. Release the mouse button when the skeleton is located in the desired location for the dialog box. The Selected Item Status window will show the exact coordinates while moving the skeleton.

15.1.3.2.2 Expanding/Shrinking a Dialog Box

To increase or decrease the size of the dialog box, use one of the eight handles (small, filled rectangles) on the boundaries. To do this, first select the dialog box. Now move the mouse pointer to a handle on the side you want

to move. The pointer will change to a small box (similar to the handle). With the mouse button depressed, drag the border in the desired direction. When you release the mouse button, the dialog box will retain its new border. You can size the box in vertical and horizontal directions simultaneously by using a corner handle.

15.1.4 Actions Bar Choices

The Application Actions Bar contains the choices:

- **F**ile
- **I**nclude
- **E**dit
- **C**ontrol
- **O**ptions
- **E**xit
- F1=Help

The bold character in each choice above is the mnemonic for the choice. The choice F1=Help will be in the rightmost position. The pop-downs for these choices contain choices as follows:

File	Choices that create, open, and save the files containing dialog boxes. There is also a choice that allows you to view and start editing existing dialog boxes.
Include	Choices that you use to create, modify, or view an include file.
Edit	Choices that allow you to perform common editing functions such as cutting and pasting dialog boxes, duplicating controls and moving groups of controls, and changing the order in which controls are accessed. There are also choices for creating a new dialog box, renaming an existing one, changing the style of controls and dialog boxes, and setting memory management flags.
Control	Choices that let you define the type of controls to be placed in the dialog box.
Options	Choices for setting Test and Translate modes, and a choice for defining the granularity of control positioning.
Exit	Choices that allow ending or resuming the application.

15.1.4.1 File Pop-down

The File Pop-down has five choices:

- New
- Open...
- Save
- Save As...
- View Dialog Box...

15.1.4.1.1 New

The function of New is to give a standard, untitled and empty resource file and clear screen to work from. If you have previously made changes to the .res or .h file image in memory, New will warn you that the file has changed and allow you to save it before clearing it from memory.

15.1.4.1.2 Open...

Open allows the editing of a dialog box from an existing .res file. When Open... is chosen, if there are unsaved changes to the current files, a message box will pop up asking if the changes should be saved before opening another file. Then the standard Open File dialog box will appear, listing the available .res files in the current directory. After a .res file is chosen, two things might happen. If the include file name is in the resource file, that include file will be opened after a message box asks for confirmation. Otherwise the Open Include dialog box listing the available .h files (include files) will be shown. The user can choose to open an include file or not. After that, the View Dialog Box dialog box listing the dialog boxes in the file will appear, and the user can choose which dialog box to view or edit.

Here is the standard Open File dialog box:

```
Open File Dialog Box
```

Current directory
: (static text) Reports what the current directory is.

Filename
: (entry) Defines the name of the resource file to open.

Available files
: (listbox) Lists the files in the current directory with the default extension .res.

15.1.4.1.3 Save

Save writes out the current .res file and .dlg file with all the dialog boxes. If the current file is untitled, Save will bring up the Save As dialog box described below. The Alt+F3 key will be an accelerator for Save. If an include file is open, its name will be saved in the resource file and, if it has changed, the user will be asked if the include file should be saved also.

15.1.4.1.4 Save As...

When Save As... is chosen, the following dialog box is displayed near the upper-left corner of the main window with the name of the current .res file filled in the Filename edit field. If the user types any extension, the Dialog Box Editor will warn that the extension is being ignored. If any symbolic definitions have changed, then the Include Save As... dialog box will also be displayed.

```
Save .res file and .dlg file Dialog Box
```

Current directory
: (static text) Reports what the current directory is.

Filename
: (entry field) Defines the name of the file to save the resource and resource definition files as.

15.1.4.1.5 View Dialog Box...

When View Dialog Box... is chosen, a dialog box is called up which displays all the dialog boxes in the current .res file. At this point, the name of the dialog box currently being edited will be highlighted, or if there is no current dialog box, the first dialog box name will be highlighted. The user can then select one of them, and this dialog box will be displayed in the editing area and will be available to be modified or tested.

```
View Dialog Box Dialog Box
```

15.1.4.2 Include Pop-down

The Include pop-down has the following choices:

- New
- Open...
- Save

- Save As...
- View Include...
- Hex Mode

This pop-down deals directly with the include files (.h files), providing a way to change the include file being edited without changing the resource file.

15.1.4.2.1 New

New clears all information copied from the current include file. If there were changes made to the image of the file which were not saved, a message box will be displayed, saying:

```
"filename.h" has changed. Save current changes?
```

before the New command is carried out.

15.1.4.2.2 Open...

Open calls up the standard Open File dialog box with a list of all the .h files (include files) in the current directory. Choosing a file, makes it the current include file for the dialog box. If there are unsaved changes to the current include file, a message box asking to save the changes will be displayed, before the Open... command is carried out.

15.1.4.2.3 Save

Save writes the current include contents to the current include file. If the current include file has no name, the Save As... dialog box will be called up.

15.1.4.2.4 Save As...

When Save As... is chosen from the pop-down, the following dialog box is displayed near the upper-left corner of the main window with the name of the current .h file filled in the Filename edit field.

```
Include Save As Dialog Box
```

15.1.4.2.5 View Include...

View Include... calls up a dialog box which allows the user to add, change, and delete ID Symbol definitions from the current include file. The current ID Symbol definitions are shown in alphabetic order in a list box. If there is no current include file, the list box is empty and the user can now add ID Symbol definitions. The definitions are not saved to an include file, unless the user issues the Save or Save As command. The dialog box for View Include... looks like this:

```
View Include Dialog Box
```

To add a control ID definition to the include file do the following. In the Symbol name text box, type the symbolic name you are giving to the control ID. In the ID Value edit box, type the number you are assigning as the ID value. If you just want what the Dialog Box Editor considers the next number, leave the ID Value field blank. Select the Add button.

To change a control ID definition, select the definition you wish to change. Now edit the symbol name and ID value in the appropriate boxes and then select the Change button.

To delete a definition, select the definition and then press the Delete button.

To change the Hex/Decimal mode of the displayed IDs, select the appropriate radio button.

15.1.4.2.6 Hex Mode

Hex mode allows the user to specify whether the Control ID values are shown in decimal or hexadecimal numbers without going through the View Include dialog box. If the ID values are shown in hexadecimal, a check mark is placed next to Hex mode in the Include pop-down.

15.1.4.3 Edit Pop-down

The Edit Pop-down has the following choices:

- **R**estore Dialog Box
- **C**ut Dialog Box
- Cop**y** Dialog Box
- **P**aste Dialog Box...
- Clear Control/Dialog Box
- **N**ew Dialog Box...

- Rename Dialog Box...
- Position relative to...
- Styles...
- Group Move
- Duplicate Control
- Resource Properties...

15.1.4.3.1 Restore Dialog Box

The Restore Dialog Box choice allows you to restore the dialog box to its previous saved state. It rereads the dialog box from the Dialog Box Editor's copy of the .res file. A message box asks for confirmation before the restoration.

15.1.4.3.2 Cut Dialog Box

This choice deletes the currently displayed dialog box and puts it in the Clipboard. (It cuts both the dialog box resource format and the bitmap format.) Individual controls cannot be cut to the Clipboard.

15.1.4.3.3 Copy Dialog Box

Copy Dialog Box puts a copy of the currently displayed dialog box (both the dialog box resource format and the bitmap format) in the Clipboard. Individual controls cannot be copied to the Clipboard, however individual controls can be duplicated with the "Duplicate Control" sub-mode.

15.1.4.3.4 Paste Dialog Box...

The Paste Dialog Box choice puts the contents of the Clipboard on the screen if the contents are in dialog box resource format. First it requests a name for the pasted dialog box and saves the current dialog box. *Note:* Only dialog boxes can be pasted; individual controls cannot be pasted from the Clipboard.

15.1.4.3.5 Clear Dialog Box/Control

This choice will read "Clear Dialog Box" if the dialog box is currently selected and "Clear Control" if a control is currently selected. It deletes the currently selected item. If it is a dialog box, a confirmation message

will be displayed saying:

```
OK to destroy currently displayed dialog box?
```

and the dialog box will be removed from the Dialog Box Editor's copy of the .res file in memory. If a control is selected, the control will just be deleted from the current copy of the dialog box being edited.

15.1.4.3.6 New Dialog Box...

New Dialog Box puts the currently displayed dialog box back into the Dialog Editor's copy of the .res file and places a new, empty dialog box on the screen. First it requests a name for the new dialog box. *Note:* This choice does not save the .res file to disk, but it does update the Dialog Box Editor's copy of the .res file in memory which can affect Restore Dialog Box.

```
New Dialog Box and Rename Dialog Box Dialog Box
```

15.1.4.3.7 Rename Dialog Box...

The Rename Dialog Box choice puts up the New Dialog Box dialog box, requesting a new name for the dialog box currently in the editing area.

15.1.4.3.8 Position Relative to...

Dialog boxes may be positioned in three ways:

1. Relative to a window
2. Relative to the screen (absolute positioning)
3. Relative to the mouse cursor

If a dialog box is positioned relative to the screen, it will always appear in the same position on the screen. If a dialog box is positioned relative to a window, it may appear at different times on different portions of the screen, but it will always appear over the same part of that window. If a dialog box is positioned relative to the mouse, the dialog box will be positioned so that a particular point on the dialog box will be under the mouse pointer when it is first displayed. This choice allows you to set the way the dialog box is positioned. The Position Relative to... choice brings up the following dialog box:

`Position Relative to Dialog Box`

The Position Relative to dialog box has three radio buttons, select one. The options are Relative to Window, Relative to Screen, and Relative to Mouse. The Window and Screen choices just set that mode and the appropriate positioning information will be remembered any time you move the dialog box. The Relative to Mouse choice produces the following dialog box and allows you to set the point which will be under the mouse pointer.

`Relative to Mouse Dialog Box`

Point to the spot you want to set and click with the mouse, the position will be displayed or you can type the (dialog) coordinates of the point you want. When done, select enter. *Note:* Since there is no way to know where the user will want to select, on or off the dialog box, the Relative to Mouse dialog box is movable.

15.1.4.3.9 Styles...

The Styles choice allows you to change the styles that govern a control or dialog box. You can also use this choice to enter or change text in a control or dialog box and to change the control's ID value and/or symbol. (If an include file was loaded, you may symbolically refer to the control's ID value. For more information on include files and ID Symbols versus ID Values, see Section 1.) Control styles dictate such things as whether a control can be grayed or whether a button is a default push button. Dialog box styles involve features such as title bars, border types and scroll bars.

To change a control style for a specific control, first select the control and choose Styles.... To change a dialog box style, first select the dialog box and choose Styles.... You will see a dialog box that relates to the control or dialog box you selected. Select the desired options. Control-style and window-style (for the dialog box) options are described in the MS OS/2 Presentation Manager Reference. If the control or dialog box has text associated with it, type the text you want to appe ar in the control in the text section. You may also type an ID value or symbol for controls. Select ENTER to end the various styles dialog boxes.

Most of the Styles dialog boxes allow you to enter text and/or an ID Value for the selected item. Usually the text is text displayed in the control, but for the dialog box, it is the text in the title bar, while for icon controls it is the name of the icon to use. For all controls, you may enter an ID Value. In this field, you may type a number (in decimal or in hexadecimal with the 0x prefix), a predefined symbol, or define a symbol by typing the symbol followed by a space then the associated number. *Note:* When you create a new control, the appropriate Style dialog box will be displayed with the next ID Value filled in. To give that value a symbol, insert the symbol followed by a space before the given number.

Button Control Styles Dialog Box

Besides the standard text and ID fields, the Button Control Styles dialog box allows you to change the type of button you have. All of the control types listed are defined by MS OS/2 Presentation Manager to be buttons.

Push Button
: is a rectangle with rounded corners and text designed to give immediate action.

Def(ault) Push Button
: is a push button with a heavy border. It is meant to be the default action on pressing the ENTER key. It should be given the ID Value 1 (ID Symbol MBID_OK).

Check Box
: is a small square with text to the right. They are usually used in groups to allow zero or more options to be selected.

Auto Check Box
: is a Check Box with which MS OS/2 Presentation Manager will maintain the checked/unchecked state. With a normal Check Box, the application is expected to check or uncheck the control when notified of the user action.

Radio Button
: is a small circle with text to the right. These buttons are used in groups to give the user a single choice from several.

3 State
: is a Check Box which can be grayed as well as marked checked or unchecked. The grayed state is typically used to show that the check box has an indeterminate state.

Auto 3 State
: is just like a 3 State, but MS OS/2 Presentation Manager maintains the visible state, toggling it from checked to unchecked and back when the user clicks in it.

Group box
: is a frame with a title on the top line, left justified. It is used to group controls together.

Edit Control Styles Dialog Box

Besides the standard ID field, the Edit Control Styles dialog box allows the text alignment and two options to be set. At run time, the application can put default text in an edit control, and the user can type text into an edit control. The text can be Left, Right, or Center Aligned. The Auto Horz. Scroll option causes the text to scroll when the edge of the field is passed. The No Hide Selection option overrides the default action of an edit field which is to highlight the selection when it receives the input focus and to hide it when it loses the focus. The Edit Control Styles dialog box has no text field because there is no text associated with an edit control.

List Box Styles Dialog Box

Besides the standard ID field, the List Box Styles dialog box allows several standard options to be set. The Notify option causes MS OS/2 Presentation Manager to notify the application whenever the user clicks or double clicks on an item in the list box. The Sort option causes the list box to sort the strings before displaying them. The Multiple Sel.(ect) option allows the user to select more than one string from the list box and to deselect a string by clicking on it again. The No Redraw option prevents the listbox from being redrawn every time changes are made. The last option, Standard, is a way of selecting/deselecting a standard set of list box options, Notify and Sort. The List Box Styles dialog box has no text field because there is no text associated with a list box control.

Static Styles Dialog Box

Besides the standard text and ID fields, the Static Styles Dialog Box allows you to select among the various styles of static controls. Static controls do not interact with the user and are just for displaying information. There are three ways of displaying text, Left, Center, and Right Aligned. The text field is for the text which will be displayed. The Icon option allows an Icon to be placed in the dialog box. The text field gives the name of the icon as given in the icon statement in the .rc file. The remaining options give various shades of rectangles or frames. These are designed to be basic building blocks for simple graphics in a dialog box (such as putting a border around some controls) and do not use the text field.

Scroll Bar Styles Dialog Box

There is no text associated with Scroll Bars.

Dialog Box Styles Dialog Box

For the Dialog Box Styles dialog box, the text field gives the Dialog Box Title. This title will be displayed in the Title Bar, and hence will only be visible if the dialog box has a title bar. The Dialog Box Styles dialog box also allows standard window style bits to be set and creates the standard frame controls. The Title Bar option gives the dialog box a title bar. The System Pop-down Box option gives the dialog box a system Pop-down box. This will only be visible if the dialog box also has a title bar. The Horz and Vert Scroll Style options give the window horizontal and vertical scroll bars. These scroll bars are part of the dialog box frame controls. The Size Border option gives the dialog box the wide size-border. The Size Box option puts a size box at the end of a scroll bar, thus the dialog box must have a scroll bar for this option to be visible. The Maximize/Minimize Box options put maximize/minimize boxes on the title bar. The Border option gives the dialog box a thin border. The Dialog Frame option gives the Dialog Box a dialog frame, a thick solid border surrounded by a thinner border. The Visible Bit sets or resets the visible style bit. This bit will be

set or reset appropriately in the .res and .dlg files, but the effect of this bit will not be displayed in the Dialog Box Editor. The visible bit determines if MS OS/2 Presentation Manager MUST show the dialog box, or if by using an accelerator key sequence the user may avoid having the box actually displayed. Generally, it is best to leave the Visible Bit check box unchecked unless you absolutely want the dialog box to be seen in all cases.

15.1.4.3.10 Group Move

Group Move toggles between normal and group move mode. Group move mode allows movement of groups of controls together. When active, Group Move is checked and "/Group" appears after the primary mode in the Dialog Box Editor's main Panel Title. Group Move and Duplicate Control Modes are mutually exclusive.

15.1.4.3.11 Duplicate Control

Duplicate Control toggles between normal and duplicate control mode. Duplicate control mode allows controls to be duplicated in all respects except for ID value. When active, Duplicate Control is checked and "/Copy" appears after the primary mode in the Dialog Box Editor's main Panel Title. Group Move and Duplicate Control modes are mutually exclusive. When the duplicated control is first placed (release of the mouse button), the appropriate Style dialog box will be displayed with the next ID value shown.

15.1.4.3.12 Resource Properties...

Since dialog boxes are resources, they have the same memory-manager flags that any resource has. The memory-manager flags determine how the code for a dialog box is treated by the application and by MS OS/2 Presentation Manager with regard to memory. You can set options to specify when a resource is to be loaded into memory, as well as whether the resource is fixed, moveable and/or discardable. The default flag settings are Moveable and Discardable on, Preload off. Memory-manager flags are set in the dialog box shown below:

```
Resource Properties Dialog Box
```

15.1.4.4 Control Pop-down

The Control Pop-down has the choices:

- Check Box
- Radio Button

- Push Button
- Group Box
- Horz. Scroll
- Vert. Scroll
- List Box
- Edit
- Text
- Frame
- Rect
- Icon

Controls in a dialog box allow the user to interact with the application. Once a border has been created for the dialog box, controls can be added by using the Control Pop-down. When one of the choices is selected from the Pop-down, the mouse pointer changes to a plus sign (+). The pointer should then be positioned where the control is to be placed. Pressing the mouse button causes the control to appear in the dialog box and the mouse pointer to change back to an arrow. If the control has text associated with it, the word "text" is included with the control when it is placed in the dialog box. Once the control is placed, the appropriate styles dialog box will come up allowing you to set the text, ID, and other features.

The choices have the following meanings:

15.1.4.4.1 Check Box

Check Box creates a check box, a small square with a label to its right. Check boxes typically are used in groups to give the user a choice of selections, any number of which can be turned on or off at a given moment.

15.1.4.4.2 Radio Button

Radio Button creates a radio button, a small circle with a label to its right. Radio buttons typically are used in groups to give the user a choice of selections, only one of which can be selected at a time.

15.1.4.4.3 Push Button

Push Button creates a push button, a small, rounded rectangle that contains a label. Push buttons are used to let the user choose an immediate action, such as canceling the dialog box. *Note:* When placing push buttons, you should leave some space between the buttons so that if one of them is

made the default push button, the wider border won't cover another border.

15.1.4.4.4 Group Box

Group Box creates a simple rectangle that has a label on its upper edge. Group boxes are used to enclose a collection or group of other controls, such as a group of radio buttons.

15.1.4.4.5 Horz. Scroll Bar

Horz. Scroll Bar creates a horizontal scroll bar. Scroll bars let the user scroll data and usually are associated with another control or window that contains text or graphics.

15.1.4.4.6 Vert. Scroll Bar

Vert. Scroll Bar creates a vertical scroll bar. Scroll bars let the user scroll data and usually are associated with another control or window that contains text or graphics.

15.1.4.4.7 List Box

List Box creates a simple rectangle that has a vertical scroll bar on its right edge. List boxes are used to display a list of strings, such as file or directory names.

15.1.4.4.8 Edit

Edit creates an edit control, a rectangle in which the user can enter and edit text. Edit controls are used both to display numbers and text and to let the user type in numbers and text.

15.1.4.4.9 Text

Text creates a static text control. Static text controls are used for Field Prompts and presenting other information such as the panel title and instructions.

15.1.4.4.10 Frame

Frame creates a rectangle that you can use to frame a control or group of controls.

15.1.4.4.11 Rectangle

Rectangle creates a filled rectangle.

15.1.4.4.12 Icon

Icon creates a rectangular space in which you can place an icon. (Do not size the icon space; icons automatically size themselves.) The text for an Icon is the name given in the icon command in the .rc file for the icon desired.

15.1.4.5 Control Manipulations

When a dialog box border is up on the screen, controls can be added to the dialog box. Once there are controls in the dialog box, they can be moved, expanded, or shrunk. To perform any of these actions, the control must be selected. This can be done by clicking the mouse on an area inside the control; the mouse pointer will be a white arrow in the areas which will select the control. When the control is selected, eight handles (small black rectangles) will appear on the boundaries of the control.

```
A Selected Text Control
```

15.1.4.5.1 Moving a Control

You can reposition a control in a dialog box either by using the mouse to drag it to a new location or by using the arrow keys for fine adjustments. To move a control, first select the control. When the mouse pointer is in the selected control, it changes to a plus sign (+). Now depress the left button and drag the control to its new location. To move a control one dialog unit at a time, use the arrow keys. In this way, you can move a control a few positions over (or up or down) without affecting its position on the other axis. This is helpful when you want to line up the controls.

15.1.4.5.2 Moving a Group of Controls

You can move more than one control in a group maintaining the relative positions of the controls. This can be useful if you decide to rearrange the layout of controls in the box and you have two or more controls that you want to keep together. To move a group of controls, first select Group Move from the Edit Pop-down then select the controls you want to move. You can select any controls you want, they don't have to be related in any way. Each control will be outlined with a gray line. The group of controls will also have a gray border around it. (If you change your mind, you can reverse a selection by clicking it with the mouse button.) Position the mouse pointer at a location inside the group border, but not inside any of

the controls' borders, as shown below (the pointer is an arrow):

```
Before Move                          After Move
```

Press the mouse button and drag the group of controls to the desired location and release the mouse button. The group of controls is placed in the new location. In the figures above, Checkbox 1 and Checkbox 3 have been selected for the group move and then are moved to the right. Checkbox 2 is not outlined and does not move. When you are done, switch back to normal Work mode by reselecting the Group Move option. There is a keyboard accelerator for group moves: hold down the Shift key whenever you press the mouse button one. This accelerator is used for selection and deselection and for dragging the group.

15.1.4.5.3 Changing a Control's Size

To increase or decrease the size of a control, use one of the eight handles (small rectangles) on the boundaries. To do this, first select the control. Now move the mouse pointer to a handle. The pointer will change to a small box, similar to the handle. Depress the left mouse button and drag the border in the desired direction. When you depress the mouse button, the small black square handles will disappear, but the square mouse pointer will remain. When the frame is the correct shape and size, release the mouse button and the control will resize to fill the frame.

```
Using the Mouse to Enlarge the Size of a Control
```

15.1.4.5.4 Duplicating a Control

To duplicate all aspects of a control except its ID, select Duplicate Control point to the control with the mouse and press the left button on the mouse. If you hold down the mouse button, you may now drag the new control. If you let up without dragging the mouse, the new control will be right on top of the old one. When you do let up on the mouse button, the appropriate Styles dialog box will appear and the ID Value will be the next available value. The new control is selected. When you finish duplicating controls, select Duplicate Control again. There is a keyboard accelerator for this command. If you hold down a Ctrl key while depressing the left mouse button while the pointer is on a control, a duplicate control will be created and selected.

15.1.4.6 Options Pop-down

The Options pop-down has the choices:

- Test Mode
- Translate Mode
- Grid...
- Order Groups...

15.1.4.6.1 Test Mode

Test Mode toggles the Dialog Box Editor between an edit mode (work or translate) and test mode. The current mode is displayed in the Dialog Box Editor's main Panel Title. Also, a check mark is placed next to the Test Mode choice in the Options pop-down when the Dialog Box Editor is in test mode. Test mode allows the dialog box to be interacted with like it was running under an actual application. The user can enter text in the edit fields, select check boxes and radio buttons, and use the TAB and DIRECTION keys to cursor around the various controls.

15.1.4.6.2 Translate Mode

Translate mode prevents any changes which will affect the interaction between the dialog box and the application. Basically it allows text and size and shape to be changed, but does not allow adding or deleting controls or changing or ID values. Translate Mode toggles between work mode, where any changes are possible, and translate mode, where only limited changes are possible. A check mark is placed next to the Translate Mode when it is active and the Dialog Box Editor's main Panel Title will display the mode as Translate.

Note: Just translating the text in the dialog box resources to another language may not be enough to translate the application to another language. All strings in the string table resource must also be translated and any static text control receiving those strings must be large enough for the translated text.

15.1.4.6.3 Grid...

The Grid choice puts up a dialog box which sets the units of the grid which determines the granularity for positioning a control when it is placed or moved. For example, when the grid is set at 20 horizontal (x) dialog units, if you select a control and try to move it to the left or right, it will move in increments of 20 units. Default settings are one unit each in both the horizontal and vertical directions.

Grid Dialog Box

15.1.4.6.4 Order Groups...

The way a dialog box reacts to the keyboard or mouse interface is based in part on the sequential order of the controls and the location of tab stops. These options are set with the Order Groups choice from the Options pop-down. Using this command, you can define the following:

The sequential order of the controls.

Which groups the controls are in, and the sequential order of the groups. (A group is a collection of controls. Within a group of controls, the user makes selections using the DIRECTION keys.)

The location of tab stops (the place where the cursor moves when the user presses the TAB key).

When the Order Groups choice is selected, the following dialog box is displayed:

Group/Control Ordering Dialog Box

The strings in the list box on the Group/Control Ordering dialog box have the following meanings. The first string in the list is "Start+of+List" padded on both sides by '+' and it allows placement of items at the start of the list. The last string in the list is "End+of+List" padded by '+' and it has a similar use. The start and end of groups are marked by "Group-Marker" padded by '-'. The strings for controls have four fields. The left-most character may be a space or an asterisk, '*', indicating a tab stop. The next field, up to the first '/', gives the text of the control. The third field, between the two '/' characters, is the ID Value. The last field is the type of control.

Changing the Order of Controls

By default, the controls you place in a dialog box receive the input focus (and thus are accessed by the user) in the order in which they were placed in the box. For example, the first control you put in the box will receive the focus first, no matter where you subsequently move it in the dialog box. To change the sequential order, you must use the Order Groups command and rearrange the controls in the list it displays.

When you rearrange the order of the controls in the Group/Control Ordering dialog box, the control statements in the .dlg file are rearranged correspondingly. Thus, the first control listed in the .dlg file is the first to receive the input focus, the second listed is the second to receive the focus, and so on.

To change the sequence in which a control gets the focus in a dialog box, choose Order Groups from the Options pop-down. From the list in the dialog box, select the control you want to move. Place the mouse pointer where you want the control to appear. Notice that as you move it, the pointer changes from an arrow to a short, horizontal bar. The bar appears only in places where you are allowed to insert the control. To insert the control, press the mouse button.

Tab Stops

Tab stops determine where the cursor will move when the user presses the TAB key. Normally, tab stops are set for individual controls or, in the case of a group, for the first control in the group. To set a tab stop, select the control at which you want to place the tab stop. Select the Tab Stop button. An asterisk appears next to the control, which indicates a tab stop has been placed. To delete a tab stop, select the control that has the tab stop. The Tab Stop button will change to read "Delete Tab". Select the Delete Tab button. The asterisk disappears.

Group Markers

To designate the beginning and end of a group, you add group markers to the list of controls in the group. (The group marker appears in the Group Order dialog box as a horizontal dashed line with the words "Group Marker" in it.) You need to place a group marker both before the first control and after the last control in a group. To add a group marker, select the control that appears just below where you want to place the group marker. Select the Group Marker button. The horizontal line indicates that the group marker has been inserted. Repeat until all markers have been placed.

To delete a group marker, select the group marker line. The Group Marker button will change to read "Delete Marker". Select the Delete Marker button.

15.1.4.7 Exit Pop-down

The Exit pop-down has the choices:

- Exit Dialog Box Editor
- Continue Dialog Box Editor

15.1.4.7.1 Exit Dialog Box Editor

Exit Dialog Box Editor will end the application. If there are unsaved changes to the resource or include files, a warning message will ask the user if the changes should be saved. The appropriate save will be done if requested by the user. The F3 key will be an accelerator for Exit Dialog Box Editor.

15.1.4.7.2 Continue Dialog Box Editor

Continue Dialog Box Editor will resume the application.

15.1.4.8 Help

Selecting F1=Help or using the keyboard accelerator, F1, will invoke user interface help for the Dialog Box Editor as described in the Help Facility for the Dialog, Font, and Icon Editors.

15.2 Font Editor Functional Specification

This document gives a functional specification of the Font Editor, an MS OS/2 Presentation Manager application. It describes the physical appearance of the application when running under MS OS/2 Presentation Manager, and also how the user interacts with the application, i.e., what the assorted commands do, and how to edit font characters.

The Font Editor lets the user edit font files to use with applications. A font file consists of a header and a collection of character bitmaps that represent the individual letters, digits, and punctuation characters that can be used to display text on a display device. Application writers who want to use fonts in their applications must add the new font files to a font resource file. *Note:* The Font Editor only handles image fonts, not outline fonts, and it does not support kerning.

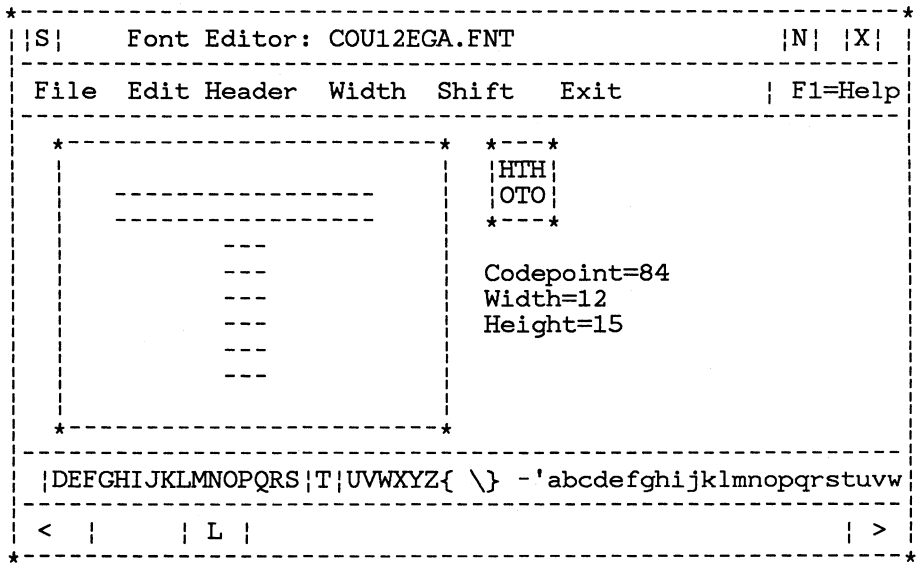
15.2.1 Application Appearance

15.2.1.1 Main Window

The main window consist of the following parts:

1. Character window

2. Character-viewing window
3. Area of text information
4. Character-selection window



L is the scroll bar slider
S is the system icon
X is the maximize icon
N is the minimize icon

Figure 15.2 Presentation Manager Font Editor

15.2.1.1.1 Character Window

The character window consists of a white box which has a grid of black lines on it. The grid cells represent individual pixels, and can be either black or white. Together, the pixels represent a single character. The character window extends from near the top of the main window down to near the top of the character-selection window.

15.2.1.1.2 Character-Viewing Window

The character-viewing window consists of a white box on a grey background, and appears to the right of the upper corner of the character window. Inside the box are two full-scale copies of the character in the character window, one above the other, with four other characters surrounding them. This window is provided so that the user has some idea of how his character will look in relation to other characters. The character-viewing window will look like one of these choices:

```
HXH     nxn     H[ H     131
OXO     oxo     O[ O     030
caps    l.c. syms    nums
```

where the middle character is the character being edited and the surrounding characters would be chosen depending if the character was a capital, a lower-case, a symbol, or a number. If the Font Editor is editing a font whose codepage it does not know about, it will treat the characters in the font as symbols.

15.2.1.1.3 Area of Text Information

Below the character-viewing window is an area which lists important information about the character. The information displayed is the character's codepoint value and its width and height in pixels.

15.2.1.1.4 Character-Selection Window

The character-selection window consists of a long horizontal box, which contains a character-selection area and a scroll bar. The character-selection area contains full-scale copies of characters in the font, and is provided to allow the user to select the current character to edit. The scroll bar allows scrolling the character-selection area so all characters in the font can be seen and selected. The character-selection window appears at the bottom of the main window, below the character window, and stretches horizontally along the bottom of the main window.

15.2.1.2 Title Bar

The window title bar will contain the text "Font Editor - filename", where filename is the name of the current font file being edited. If there is no current file loaded in the Font Editor, the title bar will contain "Font Editor - (untitled)".

15.2.1.3 Mouse Pointer Appearance

When the moused pointer is over the character window, it will appear as a pencil so that the user knows where drawing is possible. When the pointer is over selectable objects, such as the Application Action Bar (AAB), pop-down choices, and the character-selection window, it will be a black arrow with a white outline. When the pointer is over non-selectable objects, i.e, the remainder of the screen, it will appear as a white arrow with a black outline. When the Font Editor is in Add/Delete Row mode, the pointer will appear as a horizontal bar. When the Font Editor is in Add/Delete Column mode, the pointer will appear as a vertical bar.

15.2.1.4 Logo Panel

When the application first starts up, it will look in the file WIN.INI for a flag specifying whether a logo panel should be displayed, and if it should automatically continue to the program or have user controls on it to continue or quit.

If the flag specified that the logo panel should automatically continue on to the program, the line:

```
Press Enter to continue or Esc to quit.
```

would not appear on the panel.

15.2.2 Application Actions

15.2.2.1 Main Window Interaction

When a font file is loaded into the Font Editor, the "A" character is put in the character window, and the "A" character is highlighted in the character-selection area. The user can now use the mouse to edit the character in the character window, to change the current character being edited, or to choose a pop-down from the AAB.

15.2.2.1.1 Editing in the Character Window

The user edits the character in the character window by clicking the mouse while the pointer is on a pixel. If the pixel was white, it becomes black, and if the pixel was black, it becomes white. The user can also invert several pixels at once by holding the mouse button down, and dragging it over the desired pixels, and then releasing.

15.2.2.1.2 Selecting a Character in the Character-Selection Window

The user can use the scroll bar with the mouse to scroll what is visible in the character-selection area. The user can select a character to edit by moving the mouse pointer into the character-selection area of the character-selection window, and then clicking on the desired character. This causes the character to be copied to the character window, and the selection being highlighted in the character-selection area. Clicking on the arrows at the end of the scroll bar scrolls the character-selection area by one character.

15.2.2.1.3 Posting Changes to an Edited Character

The user can post his changes to the character in the character window by moving the mouse pointer over the selected character in the character-selection area and clicking the mouse button. The character-selection area is updated to show the new character. If the user clicks the mouse button while the pointer is over another character in the character-selection area, the changes to the character in the character window are still posted, and the new character is selected and copied into the character window.

When characters are edited and then posted, the character in the Font Editor's copy of the font file is changed. But this has no effect to the font file on the disk. In order to save the Font Editor's copy of the font file, the user must use the Save or Save As commands from the File pop-down. Using these commands writes out the edited font file to the disk.

15.2.2.1.4 Resizing the Main Window

When the window is resized, the contents of the window are also resized and drawn in the window to maintain full visiblity. If the window becomes too small, parts of the window are clipped.

15.2.2.2 Application Action Bar

The application action bar contains the choices:

- File
- Edit
- Header
- Width
- Shift
- Exit

The underlined character is the mnemonic for the choice. Also, the non-cursorable choice "F1=Help" will appear in the bottom rightmost position of the AAB.

15.2.2.2.1 File Pop-down

The File pop-down has the choices:

- New
- Open...
- Save
- Save As...

New

When New is chosen, if there are unsaved changes to the current font file, a warning message box will pop-up asking the user if the changes should be saved. Then the Font Editor will load in the system font. The system font is loaded as if it were a font file, filling in the header information and loading all the character bitmaps. The reason for having New use the system font is that there are many fields of information in the font's header, and it is easier for the user to have some default values to begin with than to start from scratch.

Open...

When Open... is chosen, if there are unsaved changes to the current font file, a warning message box will pop-up asking the user if the changes should be saved. Then a dialog box will be displayed near the upper left corner of the main window, prompting the user to pick a font file to load by showing the following fields:

Current directory
: (static text) Reports what the current directory is.

Filename
: (entry field) Defines the name of the font file to open

Available files
: (listbox) Lists the files in the current directory with the default extension .fnt.

Save

Save writes the Font Editor's copy of the font file out to the disk. The Alt+F3 key combination will be an accelerator for Save.

When a proportional spaced font file with a codepage the Font Editor knows about is saved, the Font Editor will do some error checking on the widths of certain characters which should have the same widths. Upon saving, the characters which should have the same widths will be checked to make sure they agree with each other. If character width mismatches are found, the following dialog box informing the user of mismatched characters will be displayed to allow the user to save the font file anyway, or return to editing characters to fix the mismatches.

The characters which should have the same widths are:

```
0123456789$
;:
.,space
+<>=
()
[ ]
{ }
OQ
il
hnu
bdpq
accented characters and their unaccented counterparts
```

Save As...

Save As brings up a dialog box which prompts the user for the name in which to save the font. It contains the following fields:

Current directory
: (static text) Reports what the current directory is.

Filename
: (entry field) Defines the name of the file in which to save.

Save As... also will do error checking for proportional spaced fonts as described under Save.

15.2.2.2.2 Edit Pop-down

The Edit pop-down has the choices:

- **Cut**
- **Copy**
- **Paste**

- Undo
- Restore

Cut copies the whole character in the character window to the Clipboard, replacing it with all white pixels.

Copy copies the whole character in the character window to the Clipboard.

Paste fills the character window with the character in the Clipboard.

Undo restores the character window to its previous state, before the last change. If the last action was to post the character into the Font Editor's copy of the font file, Undo does nothing. That is to say, Undo can only nullify the last action if the action only affected the character in the character window, and did not affect the highlighted character in the character-selection window.

Restore cancels any changes made to the edited character by recopying the character from the character-selection window to the character window.

15.2.2.2.3 Header Pop-down

The Header pop-down contains the choices:

- Naming...
- General...
- Sizes...
- Relations...

The font's header contains information about the font's size, style, weight, and other information. Since there is so much information in the header, it was impossible to present all the header information to the user in one dialog box. Thus the header information is broken into four separate dialog boxes.

Here is a complete list of all of the fields in the font's header. Following will be separate lists of what information is in each dialog box.

Typeface name
(entry field) The typeface name to which the font is designed, e.g., Times Roman.

Registry ID
(entry field) The Registry number for the font.

Character Set/Code Page
(entry field) Defines the Registered Code Page supported by the font.

First Character Code Point
(entry field) The code point of the first character in the font.

Last Character Code Point
(entry field) The code point of the last character in the font.

Default Character Code Point
(entry field) The code point which is used if a code point outside the range supported by the font is used.

Break Character Code Point
(entry field) The code point which represents the 'space' or 'break' character for this font.

Nominal Vertical Point Size
(entry field) The height of the font specified in decipoints (one 720th of an inch). This nominal size is the size for which the font is designed.

Minimum Vertical Point Size
(entry field) The minimum height to which the font may be scaled down for display.

Maximum Vertical Point Size
(entry field) The maximum height to which the font may be scaled up for display.

Weight Class
(group of radiobuttons) Indicates the visual weight (thickness of strokes) of the characters in the font. Choices are Ultra-light, Extra-light, Light, Semi-light, Medium (normal), Semi-bold, Bold, Extra-bold, and Ultra-bold.

Width Class
(group of radiobuttons) Indicates the relative aspect ratio of the character of the font in relation to the 'normal' aspect ratio for this type of font. Choices are:

- Ultra-condensed
- Extra-condensed
- Condensed
- Semi-condensed
- Medium (normal)
- Semi-expanded
- Expanded

- Extra-expanded
- Ultra-expanded

Spacing (2 radiobuttons) Indicates whether the font is fixed or proportional spaced.

Protected
(checkbox) Says whether the font is licensed or not.

Styles (group of checkboxes) Contain information concerning the nature of the font patterns, as follows:

- Italic
- Underscored
- Overstruck
- Negative Image
- Hollow Characters

Font Measurement Units
The units of measure in the font definition. Consists of:

X Unit Base
(2 radiobuttons) Describes the unit of measure base for x dimension. Tens of inches or decimeters.

Y Unit Base
(2 radiobuttons) Describes the unit of measure base for y dimension. Tens of inches or decimeters.

X Unit Value
(entry field) The number of x units of measure in the x unit base - e.g., an x and y unit of 1/1440th of an inch would be represented as 0, 0, 14400, 14400.

Y Unit Value
(entry field) The number of y units of measure in the y unit base.

Target Device Resolution - X
(entry field) The resolution in the x dimension of the device for which the font is intended, expressed as the number of device units per unit of measure.

Target Device Resolution - Y
(entry field) The resolution in the y dimension of the device.

Average Character Width
(static text) Average inter-character increment for the font; based on the "Average Character Definition Formula".

Maximum Character Increment
(entry field) The maximum inter-character increment for the font.

Maximum Baseline Extent
: (entry field) This is essentially the vertical space required by the font - i.e., the nominal inter-line gap.

Character Slope
: Defines the nominal slope for the characters of a font. The slope is defined in degrees increasing clockwise from the vertical. An Italic font is a typical example of a font with a non-zero slope. Consists of:

  Degrees (entry field) Value in the range 0-359, representing the number of degrees in the slope.

  Minutes (entry field) Value in the range 0-59, representing the number of minutes in the slope.

Inline Direction
: The direction in which the characters in the font are designed for viewing, in degrees increasing clockwise from the horizontal (left-to-right). Characters are added to a line of text along the character baseline in the inline direction. Consists of:

  Degrees (entry field) Value in the range 0-359, representing the number of degrees in the direction

  Minutes (entry field) Value in the range 0-59, representing the number of minutes in the direction

Character Rotation
: The baseline direction for which the characters in the font are designed. Consists of:

  Degrees (entry field) Value in the range 0-359, representing the number of degrees in the rotation

  Minutes (entry field) Value in the range 0-59, representing the number of minutes in the rotation

Maximum Ascender
: (entry field) The maximum height above the baseline reached by any part of any symbol in the font.

Maximum Descender
: (entry field) The maximum depth below the baseline reached by any part of any symbol in the font.

Em Height
: (entry field) The (average) height above the baseline for upper-case characters.

'x' Height
: (entry field) The (average) height above the baseline for lower-case characters.

Lower Case Ascent
: (entry field) The maximum height above the baseline reached by any part of any lower case symbol in the font.

Lower Case Descent
: (entry field) The maximum depth below the baseline reached by any part of any lower case symbol in the font.

Recommended Subscript Size
: (entry field) The recommended point size for subscripts for this font.

Recommended Subscript Position
: (entry field) The recommended baseline offset for subscripts for this font.

Recommended Superscript Size
: (entry field) The recommended point size for superscripts for this font.

Recommended Superscript Position
: (entry field) The recommended baseline offset for superscripts for this font.

Underscore Position
: (entry field) The position of the (first) underscore stroke from the baseline.

Underscore Count
: (entry field) The number of strokes used to underscore the characters in the font.

Underscore Width
: (entry field) Thickness of the underscore. (Integer + fraction).

Underscore Spacing
: (entry field) The spacing used between multiple underscores.

Strikeout Offset
: (entry field) The position of the overstrike stroke relative to the baseline.

Strikeout Thickness
: (entry field) Thickness of the overstrike stroke. (Integer + fraction).

Naming...

The Naming dialog box contains the information:

- Typeface Name
- Registry ID

- Protected (licensed)
- Character Set/Code Page
- First Character Code Point
- Last Character Code Point
- Default Character Code Point
- Break Character Code Point

General...

The General dialog box contains the information:

- Spacing (Fixed or Proportional)
- Style Options
  - Underscored
  - Italic
  - Overstruck
  - Hollow Characters
  - Negative Image
- Width Class (Ultra-condensed through Ultra-expanded)
- Weight Class (Ultra-light through Ultra-bold)

It is possible to change the font from being proportionally spaced to fixed spaced and vice-versa, merely by changing the spacing option. This basically changes nothing when going from a fixed spaced font to a proportional font - the user must do any of the character size and spacing adjustments necessary. Going from a proportional font to fixed spacing does two (destructive) things:

1. The font character definitions are all made the same width - the width of the largest character in the proportional spaced font.
2. The font spacing information is changed so that inter-character spacing is the same for all characters.

Changing a font from proportional to fixed width may cause considerable damage to a font definition because of stretching or compression on its narrow and wide characters. Therefore the Font Editor displays a warning message to notify the user of the font alteration and offers the option to cancel it.

Sizes...

The Sizes dialog box contains the following information:

- Nominal Vertical Point Size
- Minimum Vertical Point Size
- Maximum Vertical Point Size
- Font Measurement Units (X Unit Base, Y Unit Base, X Unit Value, Y Unit Value
- Target Device Resolution - X
- Target Device Resolution - Y
- Average Character Width
- Maximum Character Increment
- Maximum Baseline Extent
- Maximum Ascender
- Maximum Descender
- Em Height
- 'x' Height
- Lower Case Ascent
- Lower Case Descent

Relations...

The Relations dialog box contains the following information:

- Character Slope
- Inline Direction
- Character Rotation
- Underscore Position
- Underscore Count
- Underscore Width
- Underscore Spacing
- Strikeout Offset
- Strikeout Thickness
- Recommended Subscript Size

- Recommended Subscript Position
- Recommended Superscript Size
- Recommended Superscript Position

15.2.2.2.4 Width Pop-down

The Width pop-down has the choices:

```
1 Narrower left
2 Narrower right
3 Narrower both
4 Wider left
5 Wider right
6 Wider both
7 Set width...
```

If the font being edited is a fixed spaced font, then all of the Width pop-down choices will be grayed and non-selectable.

Narrower Left
: deletes a column from the left side of the character's bitmap.

Narrower Right
: deletes a column from the right side of the character's bitmap.

Narrower Both
: deletes a column from each side of the character's bitmap.

Wider Left
: adds a blank column to the left side of the character's bitmap.

Wider Right
: adds a blank column to the right side of the character's bitmap.

Wider Both
: adds a blank column to each side of the character's bitmap.

Note: Making characters wider than the maximum character width will bring up a message box confirming that the maximum character width will be increased.

Set Width...

Set Width... calls up the Width dialog box, which allows the user to change the width of the current character's bitmap. If the user specifies a width smaller than the current width, columns are deleted from the right side of the character's bitmap. If the user specifies a width larger than the current width, blank columns are added to the right side of the character's bitmap. If the specified size is larger than the maximum character width, a message is displayed which asks if the maximum width should be increased.

Width (entry field) Defines the width (in pixels) of the character.

15.2.2.2.5 Shift Pop-down

The Shift pop-down contains the choices:

```
1 Insert row
2 Delete row
3 Insert column
4 Delete column
```

These commands are used to insert or delete a row or column of pixels from within the character. Both commands require the user to select a row or column of pixels in the character window. When either command is chosen from the Shift pop-down, the mouse pointer changes to a horizontal bar (for insert/delete row) or a vertical bar (for insert/delete column) to signal to the user that a row or column must now be selected. The user selects the row or column by clicking the mouse pointer over the desired row or column.

Insert row
: inserts a new row of white pixels where the selected row is, pushing all other rows up or down depending on where the selected row was located. If the selected row is above the baseline, Insert pushes rows up to make room for the new row. If the selected row is below the baseline, Insert pushes rows down to make room the new row.

Delete row
: removes the selected row of pixels from the character, pushing all other rows up or down to take the removed row's place. If the selected row is above the baseline, Delete pushes rows above the selected row down towards the baseline. If the selected row is below the baseline, Delete pushes rows below the selected row up towards the baseline.

Insert column
: inserts a new column of white pixels where the selected column is, pushing all other columns left or right depending on where the selected column was located. If the selected column is on the left half of the character, Insert pushes columns to the left to make room for the new column. If the selected column is on the right half of the character, Insert pushes columns to the right to make room for the new column. If the selected column is in the exact center of the character, Insert will push columns to the right to make room for the new column.

Delete column
: removes the selected column of pixels from the character, pushing all other columns left or right to take the removed column's place. If the selected column is on the left half of the character,

Delete pushes columns to the left of the selected column towards the center of the character. If the selected column is on the right half of the character, Delete pushes columns to the right of the selected column towards the center of the character. If the selected column is in the exact center of the character, Delete will push columns to the right of the selected column towards the center.

15.2.2.2.6 Exit Pop-down

The Exit pop-down contains the choices: **E**xit Font Editor, and **C**ontinue Font Editor.

Exit Font Editor will end the application. If there are unsaved changes to the current font, a warning message box will be displayed asking the user if the changes should be saved. The F3 key will be an accelerator for Exit.

Continue Font Editor resumes the application.

15.2.3 Help

Context sensitive Help will be provided for the Font Editor as described in the document Help Facility For The Dialog, Font, and Icon Editors.

15.3 Icon Editor Functional Specification

This document gives a functional specification of the Icon Editor, a MS OS/2 Presentation Manager application. It describes the physical appearance of the application when running under MS OS/2 Presentation Manager, and also how the user interacts with the application, i.e., what the assorted commands do, and how to edit icons, pointers, and bitmaps.

The Icon Editor lets the user create customized icons, pointers, and bitmaps for use in applications. The application allows the user to work on a large-scale icon, pointer, or bitmap while displaying a full-scale replica of the work. The difference between icons, pointers, and bitmaps is as follows: *Note:* In this document, the terms hi-res, med-res, lo-res will refer to different categories of display devices. Lo-res refers to "CGA" compatible displays (640x200). Med-res refers to "EGA" compatible displays (640x350, 640x480). Hi-res refers to any displays which have a higher resolution than the "EGA" displays. Also, the following dimensions given for icons and pointers are still subject to change depending on what MS OS/2 Presentation Manager will be like.

Icons and Pointers contain 64x64 pixels in hi-res format, 32x32 pixels in med-res format, and 32x16 pixels in lo-res format. They can contain four different kinds of pixels in them: black pixels, white pixels, screen pixels, and inverse screen pixels. Screen pixels can be thought of as clear, and show the background color of whatever they are over. Inverse screen pixels show the inverse of the background color of whatever they are over. An example use of an icon is the warning symbol of the upraised hand found in some message boxes. Pointers are used by MS OS/2 Presentation Manager to show the location of the mouse on the screen.

The pixels in an icon/pointer are stored in a bitmap which is divided in two parts: the AND mask and the XOR mask. The AND mask contains the screen/non-screen color information (0 = black or white, 1 = screen or inverse screen). The XOR mask contains the invert information (0 = no invert, 1 = invert). MS OS/2 Presentation Manager draws the icon/pointer by first BITBLTing to the screen the AND mask (the result is a screen or black bitmap), and then BITBLTing to the screen the XOR mask to invert the required pixels to get white and inverse screen pixels in the bitmap. The chart below shows what the Icon Editor stores in the two bitmasks:

	Represented Color			
	Black	White	Screen	Inverse
AND mask:	0	1	1	
XOR mask:	1	0	1	

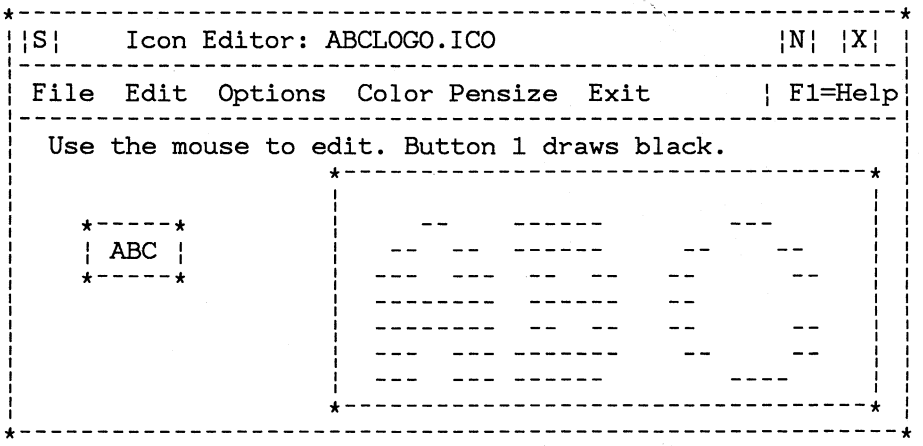
Bitmaps contain anywhere from 1x1 to 99x99 pixels. Their size is defined by the user while using the Icon Editor. They can only contain two kinds of pixels in them: black pixels or white pixels.

15.3.1 Application Appearance

15.3.1.1 Main Window

The main window consists of the following parts:

1. Editing box
2. Display box
3. Panel instructions



S is the system icon
X is the maximize icon
N is the minimize icon

Figure 15.3 Presentation Manager Icon Editor

15.3.1.1.1 Editing Box

The editing box is a large rectangular box located on the right of the main window. This box is the workspace for editing icons, pointers, and bitmaps, and its size depends on which of these is currently in use. The box is a magnified view of a small part of the screen. Each pixel in the box is many times larger than on the actual screen, so that you can see the individual pixels while doing your work.

15.3.1.1.2 Display Box

The display box is a smaller rectangular box located to the left of the editing box. It contains the type of the current figure being edited, the device category and dimensions of the figure being edited, the devic category and dimensions of the figure being viewed in the display box, and a true-scale replica of the figure being edited.

15.3.1.1.3 Panel Instructions

Below the Application Action Bar (AAB), left justified, will be instructions on what to do. What the instructions say depends on the mode. (These modes are discussed in full detail later in this document.)

Hotspot mode
: Click Button1 of the mouse in the edit box to set the hotspot.

Select mode
: Use the mouse to select a region, and then choose Cut or Copy.

Editing mode
: The instruction text will tell what color each mouse button will draw. The possible combinations are too numerous to list fully here. Some examples are:

```
"Use the mouse to Edit. Button1 draws black. Button2 draws screen."
"Use the mouse to Edit. Button1 toggles bl/wh. Button2 toggles
scr/inv."

For 1 button mouse:
"Use the mouse to Edit. Button1 draws black."
"Use the mouse to Edit. Button1 toggles scr/inv."
```

15.3.1.2 Title Bar

The window title bar will contain "Icon Editor - filename", where filename is the name of the current file being edited. If there is no current file loaded in the Icon Editor, the title bar will contain "Icon Editor - (untitled)".

15.3.1.3 Mouse Pointer Appearance

When the mouse pointer is over selectable objects, such as the AAB or pop-down choices, it will be a black arrow with a white outline. When the pointer is over non-selectable objects, i.e., the client area of the Icon Editor's window minus the editing box, it will appear as a white arrow with a black outline. When the mouse pointer is over the editing box, it will appear as one of four different pointers, depending on the mode. When the Icon Editor is in edit mode, the pointer will appear as a pencil if the pen size is 1x1, or a brush if the pen size is greater than 1x1. When the Icon Editor is in Hotspot mode, the pointer will appear as a bullseye. When the Icon Editor is in Select mode, the pointer will appear as a plus sign.

15.3.1.4 Logo Panel

When the application first starts up, it will look in the file WIN.INI for a flag specifying whether a logo panel should be displayed, and if it should automatically continue to the program or have user controls on it to continue or quit. The logo panel will appear as described in [user interface].

If the flag specified that the logo panel should automatically continue on to the program, the line "Press Enter to continue or Esc to quit." would not appear on the panel.

15.3.2 Application Actions

15.3.2.1 Main Window Interaction

When a icon, pointer, or bitmap file is loaded into the Icon Editor, a true-scale copy of the file's contents are shown in the display box, and a large-scale copy is shown in the editing box. The user can now use the mouse to edit the figure in the editing box, or choose a pop-down from the AAB.

15.3.2.1.1 Drawing in the Editing Box

The user edits (or creates) the figure in the editing box by moving the mouse pointer into the editing box and using button1 and button2 (2-button mouse) or button1 (1-button mouse) to color and erase pixels. How the mouse button(s) function depends on the figure being edited, the current pen size, and the pen color selected from the Color pop-down. The following charts define the functionality:

```
Icon/Pointer mode. Pensize>1x1.

                pen color (selected from color pop-down)
                black   white   screen  inverse
Button1 draws: black   white   screen  inverse
Button2 draws: screen  screen  screen  screen
(with 2 button mouse)

Icon/Pointer mode. Pensize=1x1. Color=Black or White.

                pixel color (of what pen is over)
                black   white   screen  inverse
Button1 draws: white   black   black   black
Button2 draws: screen  screen  inverse screen
(with 2 button mouse)

Icon/Pointer mode. Pensize=1x1. Color=Screen or Inverse screen.

                pixel color (of what pen is over)
                black   white   screen  inverse
Button1 draws: screen  screen  inverse screen
Button2 draws: white   black   black   black
(with 2 button mouse)

Bitmap mode. Pensize>1x1.
```

```
                pen color (selected from color pop-down)
                black   white
Button1 draws: black   white
Button2 draws: white   black
(with 2 button mouse)

Bitmap mode. Pensize=1x1.

                pixel color (of what pen is over)
                black   white
Button1 draws: white   black
Button2 draws: white   black
(with 2 button mouse)
```

Several pixels can be colored or erased at once by pressing the appropriate mouse button, and dragging the mouse pointer over the pixels which are to be colored or erased. The user can draw straight lines by holding down the SHIFT key while pressing a mouse button and dragging the pointer. The user can also draw straight lines by selecting the "Draw straight" choice from the Options pop-down.

15.3.2.1.2 Resizing the Main Window

When the window is resized, the editing box is resized and then the editing and display boxes are redrawn to be centered in the new window. If the window is too small, then parts of the boxes are clipped from view. The editing box will be sized no smaller than having each edit cell 2x2 screen pixels.

15.3.2.1.3 Display Device Formats

When MS OS/2 Presentation Manager loads an icon or pointer resource for an application, it scales the figure to the current system icon or pointer size. The current size depends on the display device MS OS/2 Presentation Manager is running on. This way the application writer does not have to worry about sizing his icon or pointer for different display resolutions. MS OS/2 Presentation Manager will automatically do it for him, either sizing his figure up or down depending on the size of the figure and the size of the system icon or pointer.

When MS OS/2 Presentation Manager loads a bitmap resource for an application, it does not scale the bitmap. It is the application's responsibility to stretch or compress the bitmap for its specific needs.

15.3.2.2 Application Action Bar

The application action bar contains the choices: **F**ile, **E**dit, **O**ptions, **C**olor, **P**ensize, and **E**xit. The underlined character is the mnemonic for the choice. The choice F1=Help is in the rightmost position.

15.3.2.2.1 File Pop-down

The File pop-down has four choices:

- New
- Open...
- Save
- Save As...

New

When New is chosen, if there are any unsaved changes to the current figure, a warning message box will pop-up, saying "filename" has changed. Save current changes. Then a dialog box will be displayed which prompts the user to choose a figure and its display device format. When the dialog box is entered, the Icon Editor will clear the editing box and display box of all their contents, and fill an icon or pointer with all screen pixels, and a bitmap with all white pixels.

Open...

When Open... is chosen, if there are unsaved changes to the current file, a warning message box will pop-up asking the user if the changes should be saved. Then a dialog box will be displayed near the upper left corner of the main window, showing the following fields:

Current directory
: (static text) Reports what the current directory is.

Filename
: (entry field) Defines the name of the file to open.

Available files
: (listbox) Lists the files in the current directory with the current edit default extension: .ico, .cur, or .bmp .

Save

Save writes the current figure out to the current file. If the figure is untitled, (it was newly created without being read from a file), then the Save As dialog box will be called up. The Alt+F3 key combination will be an accelerator for Save.

Save As...

When Save As... is chosen from the pop-down, a dialog box is shown near the upper left corner of the main window, showing the following fields:

Current directory
(static text) Reports what the current directory is.

Filename
(entry field) Defines the name of the file in which to save.

15.3.2.2.2 Edit Pop-down

The Edit pop-down has the choices:

- Select
- Select All
- Clear
- Cut
- Copy
- Paste
- Hotspot

Select

Select puts the Icon Editor in select mode.

In select mode, the pointer is changed to a plus sign (+), and is used to select a rectangle of pixels in the editing box. The user selects the rectangle by clicking down on a pixel, dragging the mouse to another pixel, and releasing the mouse button. While the user is dragging the mouse, he will see the frame of a rectangle displayed from where the mouse pointer is to the pixel the mouse pointer was clicked down on. You can change your selection by doing the click-drag-release actions again.

Exit select mode by either choosing the Select choice again, or choosing Clear, Cut or Copy from the Edit pop-down.

Select All

Select All also puts the Icon Editor in select mode but the entire figure is automatically selected. You can change the selection by click-drag-release actions as specified for Select.

Clear

Clear erases the selected rectangle of pixels by replacing them with white pixels in bitmap mode, or with screen pixels in icon or pointer mode. The Icon Editor returns to edit mode after executing this command. The Clear command does not affect the Clipboard contents.

Cut

Cut will copy the selected rectangle of pixels to the Clipboard, and will replace the pixels with white pixels in bitmap mode, and screen pixels in icon and pointer mode.

In bitmap mode the Clipboard receives one bitmap. In cursor or icon modes the Clipboard receives two objects: a black and white bitmap, where white represents the figure's white and screen pixels, and black represents the figure black and inverse screen pixels; and an object of a format defined privately by the Icon editor, which maintains all the figure's information. The privately defined object is for use only by the Icon Editor.

After the Cut command has been done, the Icon Editor will be taken out of select mode.

Copy

Copy will copy the selected rectangle of pixels to the Clipboard. The Clipboard new contents will be as specified in the Cut command section. After the Copy command has been done, the Icon Editor will be taken out of select mode.

Paste

Paste allows the user to copy pixels from the Clipboard to the currently edited figure. If the currently edited figure is a pointer or an icon and the Clipboard contains a figure defined in the Icon Editor private format, the Clipboard contents will be scaled to the current display device resolution. Otherwise the Clipboard contents will not be scaled.

If both Clipboard contents and figure are either icon or pointer, screen and inverse screen information is preserved, otherwise only black and white pixels are pasted using the Clipboard bitmap. For a summary refer to the following table:

	pasting bitmap into bitmap	bitmap into icon/ptr	icon/ptr into bitmap	icon/ptr into icon/ptr
Clipboard object used	bitmap	bitmap	bitmap	icon/ptr
Scaling	no	no	no	yes
Colors	Black/White	Black/White	Black/White	Black/White and screen/inverse

How the actual pasting is done depends on how the Clipboard contents fit into the figure being edited. Let "larger" mean that a figure exceeds the size of the currently edited figure in either dimension. If a figure is identical in size in both dimensions to the currently edited figure, then it is the

"same size". Otherwise the figure is "smaller".

If the Clipboard figure is smaller, the frame of a rectangle representing the pixels in the Clipboard will be displayed in the editing box. The user can then move the frame to a location in the editing box by clicking inside the rectangle, dragging the mouse, and then releasing at the desired location. The editing box will then be repainted to incorporate the pixels from the Clipboard.

If the Clipboard figure is larger than the current figure size (if you are editing a pointer or an icon) or than the maximum bitmap size (if you are editing a bitmap), a warning message will be displayed notifying the user that part of the Clipboard contents will be clipped, and Paste will only copy the pixels in the rectangle frame which intersect the editing box.

If the Clipboard figure is larger than the currently edited bitmap but smaller than the bitmap maximum size, a warning message will be displayed notifying the user that the current bitmap size will be increased to the Clipboard figure size and that the later will replace entirely the previously edited bitmap.

If the Clipboard figure has the same size, a warning message will be displayed and, given the user confirmation, the Clipboard figure will replace entirely the previous edited picture.

Hotspot

In an icon, the hotspot can be used by applications to determine where the icon is on the screen. In a pointer, the hotspot is the pixel from which MS OS/2 Presentation Manager will take the pointer's current screen coordinates. Bitmaps do not have hotspots.

The Hotspot choice is only enabled if editing an icon or pointer; if editing a bitmap, Hotspot is grayed. When Hotspot is chosen, a checkmark is placed next to it, information about the current hotspot location appears in the display box (see picture be low), and the mouse pointer changes to a bullseye. The hotspot is set by moving the mouse pointer to the location in the icon or pointer, (in the editing box), where the hotspot is desired, and clicking the mouse.

Only one hotspot is allowed, so clicking the bullseye pointer elsewhere will reset the hotspot location. If the user does not set the hotspot, the default location is the center of the icon or pointer. While in hotspot mode, Select, Cut, Copy, and Paste are grayed. To leave hotspot mode, the user must again select Hotspot from the pop-down; the checkmark is removed, the other pop-down choices enabled, and the location information will disappear.

15.3.2.2.3 Options Pop-down

The Options pop-down has the choices: **G**rid, **D**raw straight, **B**lack, **W**hite, **D**ark (blue/gray), Light (blue/gray), **L**o-res, **M**ed-res, and **H**i-res.

All the choices in the Options pop-down affect what the user sees while editing his figure, but do not affect what is actually stored in the figure.

Grid

When Grid is chosen, a checkmark is placed by it, and a grid of lines is displayed over the editing box. Each grid cell represents one pixel in the figure's bitmap. If the background color is Black, the grid is made up of white lines. If the background is anything other than Black, the grid is made up of black lines. Choosing Grid again will remove the checkmark and remove the grid on the editing box.

Draw Straight

When Draw straight is chosen, a checkmark is placed by it, and the Icon Editor will now draw/erase straight lines in the editing box. If the user clicks on a pixel in the editing box and then drags his mouse horizontally, the Icon Editor will force drawing a horizontal line, even if the user deviates from the horizontal row. Similarly, if the user clicks down on a pixel and then drags his mouse vertically, the Icon Editor will draw a vertical line. Once the user releases the mouse button, the current direction which was being forced is no longer forced. To leave the Draw straight mode, the user must choose the Draw straight choice again. The user can also draw straight lines by holding down the SHIFT key while pressing a mouse button and dragging the pointer.

Black, White, Dark (blue/gray), Light (blue/gray)

These choices will have the text "Background View:" above them on the Options pop-down. If the user is running on a color display, the third and fourth choices will be Dark blue and Light blue. If the user is running on a monochrome display, the third and fourth choices will be Dark gray and Light gray.

The background color is provided to allow the user to see what his icon, pointer, or bitmap will look like over a variety of different screen colors. It is for viewing purposes only, it does not affect what is stored inside the icon, pointer, or bitmap. The background color does not actually fill in pixels in the figure; it is seen through screen pixels and its inverse is seen through inverse screen pixels. When one of the color choices is chosen, a checkmark is placed by it and the editing box and display box are redisplayed with the new background color.

Lo-res, Med-res, Hi-res

These choices will have the text "Display Version View:" above them on the Options pop-down. If the user is editing a dependent bitmap, these choices will be grayed.

These choices decide which display version of the current figure to show in the display box. If the user is editing either an icon or pointer, the Icon Editor will use the same algorithm MS OS/2 Presentation Manager will use to either compress or stretch the figure to be the same size as the chosen display version's system icon or pointer. For independent bitmaps, the Icon Editor will use the same algorithm MS OS/2 Presentation Manager will use to scale the bitmap up or down depending on the chosen display version. (The scaling factors have not been decided at the time of writing). If the user is editing a dependent bitmap, the display version will be ignored, and the figure in the display box will be exactly the number of pixels the user specified for the bitmap.

15.3.2.2.4 Color Pop-down

The Color pop-down has the choices: **B**lack, **W**hite, **S**creen, and **I**nverse screen.

In pointer and icon mode, all pen color choices are enabled. In bitmap mode, Screen and Inverse screen are grayed because bitmaps can only contain black or white pixels.

The pen color choice always refers to Button1 of the mouse. If the user is using a 2-button mouse, the color of Button2 will be based on the color selected for Button1. The panel instructions will describe which colors are assigned to each mouse button.

When one of the choices is chosen, a checkmark is placed next to it and the current pen color is set to the new choice.

15.3.2.2.5 Pensize Pop-down

The Pensize pop-down has the choices: **S**mall (1x1), **M**edium (3x3), **L**arge (5x5), and **E**xtra large (7x7).

When one of the choices is chosen, a checkmark is placed next to it and the current pen size is set to the new choice. Now drawing in the editing box will fill a region of pixels equal to the new size.

When the pen size is greater than one pixel, the region filled will be located around the tip of the pen pointer, i.e., the pen pointer's tip will be in the center of the 3x3 region (medium size) or 5x5 region (large size) or 7x7 region (extra large size).

15.3.2.2.6 Exit Pop-down

The Exit pop-down contains the choices: **E**xit Icon Editor, and **C**ontinue Icon Editor.

Exit Icon Editor will end the application. If there are unsaved changes to the current figure, a warning message box will be displayed asking the user if the changes should be saved. The F3 key will be an accelerator for Exit.

Continue Icon Editor resumes the application.

15.3.3 Help

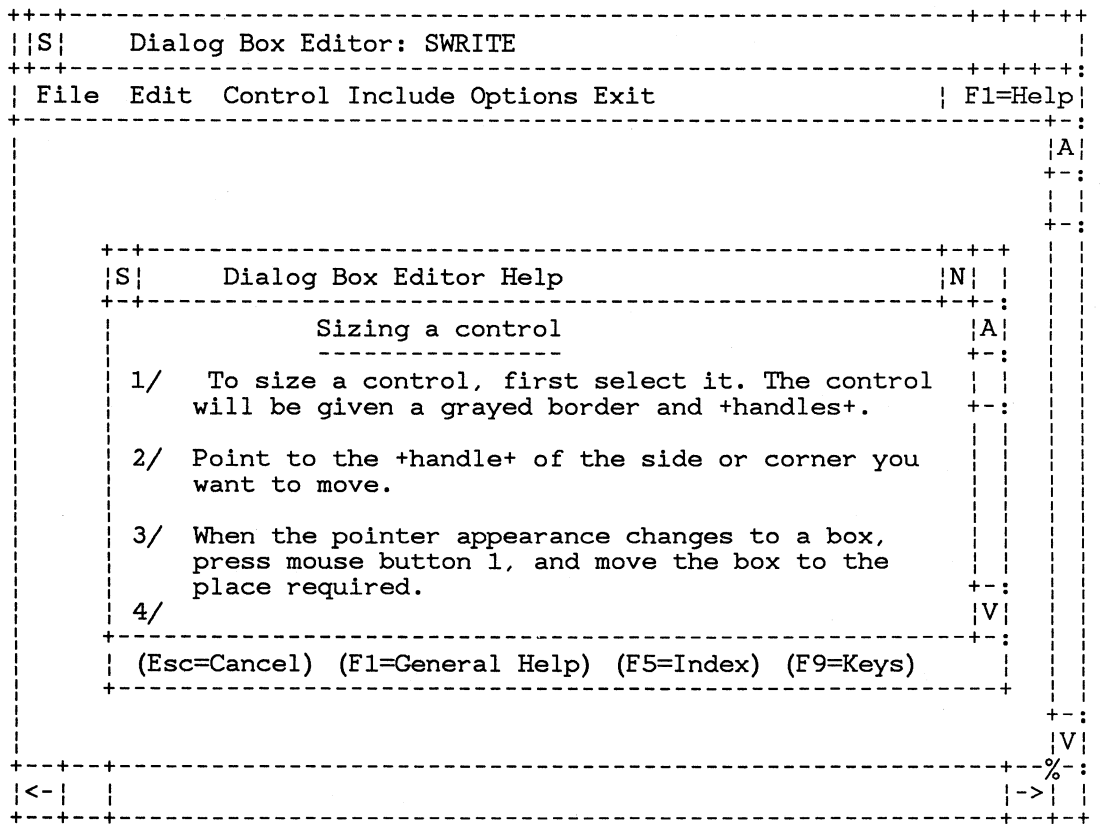
Context sensitive Help will be provided for the Icon Editor, as defined in the document Help Facility for the Dialog, Font, and Icon Editors.

15.4 Help Facility for the Dialog, Font, and Icon Editors

The purpose of Help is to provide information to the user which aids in the operation of an application. When the user requests Help, information regarding the item selected in the current context is displayed. The user can also request an index of available Help topics, request General Help, or request information on the functions assigned to keys.

The appearance and function of the Help Facility for the Dialog, Font and Icon Editors is the same as for the Shell.

Here is a picture of what the Help window might look like for the Editors:



A sample help window

15.5 Resource (.res) File Specification

The format for the .res file is as follows:

```
(/TYPE NAME FLAGS SIZE BYTES/)+
```

Where:

TYPE is either a null-terminated string or an ordinal, in which case the first byte is 0xFF followed by an INT which is the ordinal.

```
/* Predefined resource types */
#define RT_POINTER      MAKEINTRESOURCE( 1 )
#define RT_BITMAP       MAKEINTRESOURCE( 2 )
#define RT_ICON         MAKEINTRESOURCE( 3 )
#define RT_MENU         MAKEINTRESOURCE( 4 )
#define RT_DIALOG       MAKEINTRESOURCE( 5 )
#define RT_STRING       MAKEINTRESOURCE( 6 )
#define RT_FONTDIR      MAKEINTRESOURCE( 7 )
#define RT_FONT         MAKEINTRESOURCE( 8 )
#define RT_ACCELTABLE   MAKEINTRESOURCE( 9 )
#define RT_DLGINCLUDE   MAKEINTRESOURCE(10 )
```

NAME is the same format as TYPE. There are no predefined ordinals.

FLAGS is an unsigned value containing the memory manager flags:

```
#define NSTYPE      0x0007   /* Segment type mask */
#define NSCODE      0x0000   /* Code segment */
#define NSDATA      0x0001   /* Data segment */
#define NSITER      0x0008   /* Iterated segment flag */
#define NSMOVE      0x0010   /* Movable segment flag */
#define NSPURE      0x0020   /* Pure segment flag */
#define NSPRELOAD   0x0040   /* Preload segment flag */
#define NSEXRD      0x0080   /* Execute-only (code segment),
                             *  or read-only (data segment) */
#define NSRELOC     0x0100   /* Segment has relocations */
#define NSDEBUG     0x0200   /* Segment has debug info */
#define NSDPL       0x0CC0   /* 286 DPL bits */
#define NSDISCARD   0x1000   /* Discard bit for segment */
```

SIZE is a LONG value telling how many bytes follow in the resource.

BYTES is the stream of bytes that makes up the resource.

Any number of resources can appear one after another in the .res file.

15.6 Resource Script File Specification

15.6.1 Resource Script File

The resource script file defines the names and attributes of the resources to be added to the application's executable file. The file consists of one or more resource statements that define the resource type and original file.

The following is a list of the resource statements:

Single-line statements

- POINTER
- ICON
- BITMAP
- FONT
- DLGINCLUDE

User-defined resources

Multiple-line statements

- STRINGTABLE
- ACCELTABLE
- MENU
- DLGTEMPLATE
- WINDOWTEMPLATE

Directives

- #include
- #define
- #undef
- #ifdef
- #ifndef
- #if
- #elif
- #else
- #endif

The following sections describe these statements in detail.

15.6.1.1 Single Line Statements

The single line statements define resources that are contained in a single file, such as pointers, icons, and fonts. The statements associate the filename of the resource with an identifying name or number. The resource is added to the executable file when the application is created, and can be extracted during execution by referring to the name or number.

The general form for all single line statements is:

```
resource-type nameID (.load-option.) (.mem-option.) filename
```

nameID is either a unique name or an integer number identifying the resource. For a FONT resource, the nameID must be a number; it cannot be a name.

resource-type is one of the following keywords, specifying the type of resource to be loaded:

Keyword	Resource Type
POINTER	A pointer resource is a bitmap defining the shape of the mouse pointer on the display screen.
ICON	An icon resource is a bitmap defining the shape of the icon to be used for a given application.
BITMAP	A bitmap resource is a custom bitmap that an application intends to use in its screen display or as an item in a menu.
FONT	A font resource is simply a file containing a font. The format of a font file is defined in Appendix C.
DLGINCLUDE	This statement tells the dialog box editor which file to use as an include file for the dialog boxes in the resource file. The NameId is not applicable.

load-option is an optional keyword specifying when the resource is to be loaded. It must be one of the following:

PRELOAD
: Resource is loaded immediately

LOADONCALL
: Resource is loaded when called

The default is LOADONCALL.

The mem-option consists of the following keyword or keywords, specifying whether the resource is fixed or moveable and whether it is discardable:

FIXED Resource remains at a fixed memory location

MOVEABLE
: Resource can be moved if necessary to compact memory

DISCARDABLE
: Resource can be discarded if no longer needed

The default is MOVEABLE and DISCARDABLE for POINTER, ICON, and FONT resources. The default for BITMAP resources is MOVEABLE.

filename is an ASCII string specifying the MS OS/2 filename of the file containing the resource. A full pathname must be given if the file is not in the current working directory.

Examples:

```
POINTER pointer point.cur
POINTER pointer DISCARDABLE point.cur
POINTER 10 custom.cur

ICON desk desk.ico
ICON desk DISCARDABLE desk.ico
ICON 11 custom.ico

BITMAP disk disk.bmp
BITMAP disk DISCARDABLE disk.bmp
BITMAP 12 custom.bmp

FONT 5 CMROMAN.FON
```

15.6.1.2 User-Defined Resources

An application can also define its own resource. The resource can be any data that the application intends to use. A user-defined resource statement has the form:

```
RESOURCE  typeID nameID  (.load-option.) (.mem-option.) filename
```

typeID is either a unique name or an integer number identifying the resource type. If a number is given, it must be greater than 255. The type numbers 1 through 255 are reserved for existing and future predefined resource types.

nameID is either a unique name or an integer number identifying the resource.

load-option is an optional keyword specifying when the resource is to be loaded. It must be one of the following:

PRELOAD
: Resource is loaded immediately

LOADONCALL
: Resource is loaded when called

The default is LOADONCALL.

mem-option consists of the following keyword or keywords, specifying whether the resource is fixed or moveable and whether it is discardable:

FIXED Resource remains at a fixed memory location

MOVEABLE
Resource can be moved if necessary to compact memory

DISCARDABLE
Resource can be discarded if no longer needed

The default is MOVEABLE.

filename is an ASCII string specifying the MS OS/2 filename of the file containing the pointer bitmap. A full pathname must be given if the file is not in the current working directory.

Example:

```
RESOURCE MYRES    array    data.res
RESOURCE 300      14       custom.res
```

15.6.1.3 Codepage Flagging

The following resource types each have a codepage associated with them:

- STRINGTABLE
- ACCELTABLE
- MENU
- WINDOWTEMPLATE and DLGTEMPLATE

The codepage is encoded in the resource, and the data in the resource is assumed to be in the specified codepage. However, no checking is performed.

The following codepages may be specified:

- 437
- 850
- 860
- 863
- 865

If the codepage is not specified, then codepage 850 is assumed.

15.6.1.4 STRINGTABLE Statement

The STRINGTABLE statement defines one or more more string resources for an application. String resources are simply null-terminated ASCII strings that can be loaded when needed from the executable file, using the LoadString function.

The STRINGTABLE statement has the form:

```
STRINGTABLE (.load-option.)(.mem-option.) (.codepageid.)
BEGIN
string-definitions
END
```

where string-definitions are one or more ASCII strings, enclosed in double quotation marks and preceded by an identifier. The identifier must be an integer.

load-option is an optional keyword specifying when the resource is to be loaded. It must be one of the following:

PRELOAD
: Resource is loaded immediately

LOADONCALL
: Resource is loaded when called

The default is LOADONCALL.

The optional mem-option consists of the following keyword or keywords, specifying whether the resource is fixed or moveable and whether it is discardable:

FIXED Resource remains at a fixed memory location

MOVEABLE
: Resource can be moved if necessary to compact memory

DISCARDABLE
: Resource can be discarded if no longer needed

The default is MOVEABLE and DISCARDABLE.

Example:

```
#define IDS_HELLO     1
#define IDS_GOODBYE   2

STRINGTABLE
BEGIN
    IDS_HELLO,    "Hello"
    IDS_GOODBYE,  "Goodbye"
END
```

Note: In addition to the STRINGTABLE keyword, there is an equivalent MESSAGETABLE keyword. It is identical to the STRINGTABLE except that a different resource ID value is generated on compilation. The MESSAGETABLE keyword is mainly used for MS OS/2 Presentation Manager error messages and need not be used by applications.

15.6.1.5 Accelerator Tables

The ACCELTABLE statement defines a table of accelerator keys for an application.

An accelerator is a keystroke defined by the application to give the user a quick way to perform a task. The TranslateAccelerator function is used to translate accelerator messages from the application queue into WM_COMMAND, WM_HELP or WM_SYSCOMMAND messages.

The ACCELTABLE statement has the form:

```
ACCELTABLE  <id>     <memory mgr flags> <codepageid>
BEGIN
  <keyval>, <cmd>, <acceloption , acceloption >
  ...
END
```

id is the resource id.

Codepageid

keyval is the accelerator character code. This can either be a constant, or a quoted character. If it is a quoted character, then the CHAR acceloption is assumed. If the quoted character is preceded with an up-arrow character, then a control character is specified.

cmd is the value of the WM_COMMAND, WM_HELP or WM_SYSCOMMAND message generated from the accelerator for the indicated key.

acceloption
defines the kind of accelerator.

The VIRTUALKEY, SCANCODE, and CHAR acceloptions specify the type of message that will match the accelerator. Only one of these options may be specified per accelerator.

The acceloptions SHIFT, CONTROL, and ALT, cause a match of the accelerator only if the corresponding key is down.

If there are two accelerators that use the same key with different SHIFT, CONTROL, or ALT options, the more restrictive accelerator should be specified first in the table. For example, Shift-Enter should be placed before Enter.

The SYSCOMMAND acceloption causes the keystroke to be passed to the application as a WM_SYSCOMMAND message. If it is not specified, a WM_COMMAND message is used.

The HELP acceloption causes the keystroke to be passed to the application as a WM_HELP message. If it is not specified, a WM_COMMAND message is used.

Note that the AF_XXX form of these constants can also be used. These can be OR'ed together, e.g., AF_CHAR | AF_HELP. (See the section on accelerator tables).

Example:

```
ACCELTABLE MainAcc
BEGIN
     "S", 101, CONTROL
     "G", 102, CONTROL
END
```

This would be used to generate WM_COMMAND messages with values of 101 and 102 from Control-S and Control-G. This might be used in conjunction with menu options for Saving and Getting files, for example.

15.6.1.6 MENU Statement

The MENU statement defines the contents of a menu resource. A menu resource is a collection of information that defines the appearance and function of an application menu. A menu is a special input tool that lets a user select commands from a list of command names.

The MENU statement has the form:

```
MENU     <menuID>     <load option> <mem-option> <codepageid>
BEGIN
    MENUITEM     "string", <cmd>, <flags>
    if (<flags> includes MIS_POPOP)
    BEGIN
        MENUITEM
    END
END
```

menuID is a name or number used to identify the menu resource.

load-option
is an optional keyword specifying when the resource is to be loaded. It must be one of the following:

PRELOAD
Resource is loaded immediately

LOADONCALL
Resource is loaded when called
The default is LOADONCALL.

mem-option
is optional. It consists of the following keyword or keywords, specifying whether the resource is fixed or moveable and whether it is discardable:

FIXED Resource remains at a fixed memory location

MOVEABLE
Resource can be moved if necessary to compact memory

DISCARDABLE
Resource can be discarded if no longer needed

Codepageid

MENUITEM
is a special resource statements used to define the items in the menu. These are discussed in more detail in the next section.

Example

The following is an example of a complete MENU statement:

```
MENU sample
BEGIN
        MENUITEM "Alpha", 100, MIS_TEXT
        MENUITEM "Beta", 101, MIS_TEXT¦MIS_SUBMENU
        BEGIN
                MENUITEM "Item 1", 200  MIS_TEXT
                MENUITEM "Item 2", 201, MIS_TEXT¦MIA_CHECKED
        END
END
```

15.6.1.6.1 Menu Item Definition Statements

MENUITEM statements are used in the item-definition section of a MENU statement to define the names and attributes of the actual menu items. Any number of statements can be given; each defines a unique item. The order of the statements defines the order of the menu items. *Note:* The MENUITEM statements can only be used within an item-definition section of a MENU statement.

```
MENUITEM    "string", <cmd>, <flags>
```

string is an ASCII string, enclosed in double quotation marks, specifying the name of the menu item.

The string can contain the escape characters \t and \a. The \t character inserts a tab in the string when displayed and is used to align text in columns. Tab characters should be used only in popup menus, not in menu bars. The \a character right-justifies all text that follows it.

The Tab and Right Justify functions work as follows:

- When formatting a Pull-down menu, two lengths are defined:
  1. The maximum length of any string before a \t or \a.(First Length)
  2. The maximum length of any string following a \t or \a.
- The total width of the menu is equal to the sum of the two lengths plus the width of a blank character.
- A string with a \t embedded is formatted so that all the text prior to the \t is presented left justified. The text following the \t is presented starting at an x position equal to the start of the prior text plus the First Length plus the width of a blank character.
- A string with a \a embedded is formatted like a string with a \t embedded, except that the text following the \t is positioned so that it is right justified.

After compilation, the menu resource contains the \t and \a indicators as control characters. For example, the \t is stored as 0x09.

One of the main uses of these functions is to present accelerator keys which are equivalent to the selection of a given menu item. For example a menu might appear as:

```
Insert Text Ctrl+I
Delete Text Ctrl+D
Copy Text   Ctrl+C
Search Text Ctrl+S
```

The Ctrl+* text is a description of the Accelerator Key for the menu item and should be coded in the menu item text string with a preceding \t. In this way, it is not necessary to calculate widths of text in order to get correct appearance of the menu.

To insert a double quote character (") in the text, use two double quote characters ("").

The string can also contain a tilde character indicating that the following character is used as a mnemonic character for the item. A full explanation of the use of mnemonics is given in the section dealing with Menus.

If <flags> does not contain MIS_TEXT, the string is ignored but must still be specified. An empty string ("") should be specified in this case.

cmd — is an integer number. This number is used as the command value in the WM_COMMAND message (or WM_SYSCOMMAND message, if MIS_SYSCOMMAND is specified in <flags>), which is sent to the owner window when the user selects the menu item. Hence it identifies the selection made and should be unique within one menu definition.

This is the same as the *idItem* field in the Menu Template for an Item.

flags — are one or more menu options defined by the MIS_ and MIA_ constants, ORed together with the ¦ operator. These constants and their meaning are fully defined in the section on Menu Controls.

Examples:

```
MENUITEM  "Alpha", 1, MIS_TEXT¦MIA_ENABLED¦MIA_CHECKED
MENUITEM  "Beta", 2, MIS_TEXT
```

15.6.1.6.2 Pull-down Menus/Submenus

As well as simple items, a menu definition can contain the definition of a Pull-down Menu or Submenu. The main menu appears as a horizontal bar of items at the top of the window to which it relates. Pull-down menus appear as vertical lists running downwards from an item in the main menu, which only become visible as the result of a selection on the item in the main menu.

The definition of a Pull-down menu is very similar to that of the main menu - it consists of a list of MENUITEM statements. It is introduced by an item in the main menu which has the MIS_SUBMENU constant set.

Example:

```
MENU chem
BEGIN

MENUITEM "elements", 2, MIS_TEXT¦MIS_SUBMENU
BEGIN
        MENUITEM "Oxygen", 200  MIS_TEXT
        MENUITEM "Carbon", 201, MIS_TEXT¦MIA_CHECKED
        MENUITEM "Hydrogen", 202, MIS_TEXT
END

MENUITEM "Compounds", 2, MIS_TEXT¦MIS_SUBMENU
BEGIN
        MENUITEM "Glucose", 301, MIS_TEXT
        MENUITEM "Sucrose", 302, MIS_TEXT¦MIA_CHECKED
        MENUITEM "Lactose", 303, MIS_TEXT¦MIS_BREAK
        MENUITEM "Fructose", 304, MIS_TEXT
END

END
```

15.6.1.6.3 Separator Menu Item

There is a special form of the MENUITEM statement which is used to create a horizontal dividing bar between two active menu items in a Pull-down menu. The Separator item is itself inactive and has no text associated with it nor a cmd value.

Example:

```
MENUITEM  "Roman", 206, MIS_TEXT
MENUITEM  SEPARATOR
MENUITEM  "20 Point", 301, MIS_TEXT
```

15.6.1.7 DIALOG and WINDOW Templates

DLGTEMPLATE and WINDOWTEMPLATE statements are used by an application to create predefined window and dialog resource templates.

The DLGTEMPLATE and WINDOWTEMPLATE statements are treated identically by the resource compiler and have the following format:

```
(DLGTEMPLATE ¦ WINDOWTEMPLATE) resourceID loadoption memoption codepageid
(BEGIN ¦ (/)
    Single DIALOG, CONTROL, or WINDOW statement
(END ¦ /))
```

The parts of the DLGTEMPLATE and WINDOWTEMPLATE statements are described below.

Purpose This statement marks the beginning of a window template. It defines the name of the dialog box window, and its memory and load options.

Parameters

resourceID is either a unique name or an integer number identifying the resource.

load-option is an optional keyword specifying when the resource is to be loaded. It must be one of the following:

Option	Meaning
PRELOAD	Resource is loaded immediately
LOADONCALL	Resource is loaded when called

The default is LOADONCALL.

The optional mem-option consists of the following keyword or keywords, specifying whether the resource is fixed or moveable and whether it is discardable:

Option	Meaning
FIXED	Resource remains at a fixed memory location
MOVEABLE	Resource can be moved if necessary to compact memory
DISCARDABLE	Resource can be discarded if no longer needed

The default is MOVEABLE.

Codepageid

Alternatively, "(/" can be used in place of BEGIN and "/)" in place of END.

The DLGTEMPLATE and WINDOWTEMPLATE keywords are synonyms.

The DIALOG statement defines a window of class WC_DIALOG that can be used by an application to create dialog boxes.

The DIALOG statement has the format:

```
DIALOG      text, id, x, y, width, height, (., style.)
(.CTLDATA    (MENU ¦ data, data, ....).)
(.PRESPARAMS data, data, .....)
BEGIN
    one or more DIALOG, CONTROL, WINDOW statements
END
```

The parts of the DIALOG statement are described below.

Purpose This statement marks the beginning of a DIALOG statement. It defines the box's starting location on the display screen, its width, its height, and any extra style bits.

Parameters

text is a string that is displayed in the title bar control, if it exists.

x and y are integer numbers specifying the x and y coordinates on the display screen of the lower left corner of the dialog box. x and y are in dialog coordinates. The exact meaning of the coordinates depends on the style defined by the style argument. For normal dialog boxes, the coordinates are relative to the origin of the parent window. For DS_SCREENALIGN style boxes, the coordinates are relative to the origin of the display screen. With DS_MOUSEALIGN, the coordinates are relative to the position of the mouse pointer at the time the dialog box is created.

width and height are integer numbers specifying the width and height of the box. The width units are 1/4 the width of a character; the height units are 1/8 the height of a character.

style is any additional window styles, dialog styles, or frame styles.

The WINDOW and CONTROL statements have the format:

```
(CONTROL ¦ WINDOW)  text, id, x, y, width, height, class (., style.)
(.CTLDATA    (MENU ¦ data, data, ....).)
(.PRESPARAMS data, data, .....)
BEGIN
    one or more DIALOG, CONTROL, WINDOW statements
END
```

Note: The WINDOW and CONTROL keywords are synonyms.

The BEGIN-END pair can be deleted if there are no child dialog, control or window statements.

The DIALOG, CONTROL and WINDOW statements between the BEGIN and END statements are defined as child windows. The template format is fully recursive - the DIALOG/CONTROL/WINDOW statements between the BEGIN/END may also have BEGIN/END blocks.

The optional CTLDATA statement is used to define control data for the control. The CTLDATA statement can take one of the following forms:

```
CTLDATA 0xnn,0xnn,.... Hexadecimal byte values
CTLDATA "string"         String constant
CTLDATA nn,nn,nn,... Decimal word date
CTLDATA MENU             Menu Template as control data
BEGIN
  ...
END
```

In addition to hex or decimal data, the CTLDATA statement may be followed by the MENU keyword, followed by a menu template in a BEGIN/END block. This creates a menu template as the window's control data.

The optional PRESPARAMS statement is used to define presentation parameters. The syntax of the PRESPARAMS statement is similar to the CTLDATA statement:

```
PRESPARAMS  PP_*, nn, PP_*, nn,..... Pairs of PP_* values and
                                        parameter values.
```

Left and right curly braces are synonyms for BEGIN and END.

In addition to the normal CONTROL statement, there exist special statements for commonly used controls such as pushbuttons and edit controls. These have the same format as the normal CONTROL statement, except that their STYLE and CLASS statements are implied.

Examples

The following is a complete example of a DIALOG statement.

```
#include "windows.h"

DLGTEMPLATE errmess BEGIN
    DIALOG  "Disk Error", 100, 10, 10, 300, 110
    BEGIN
        CTEXT "Select One:", 1, 10, 80, 280, 12
        RADIOBUTTON "Retry", 2, 75, 50, 60, 12
        RADIOBUTTON "Abort", 3, 75, 30, 60, 12
        RADIOBUTTON "Ignore", 4, 75, 10, 60, 12
    END
END
```

This is an example of a WINDOWTEMPLATE statement that is used to define a specific kind of window frame. Calling WinLoadDlg with this resource will automatically create the frame window, the frame controls, and the client window (of class MyClientClass).

```
WINDOWTEMPLATE wind1
BEGIN
    FRAME "My Window", 1, 10, 10, 320, 130, FS_STANDARD | FS_VERTSCROLL
    BEGIN
        WINDOW "", FID_CLIENT, 0, 0, 0, 0, "MyClientClass", style
    END
END
```

This example creates a resource template for a modeless dialog box identified by the constant "modeless1". It includes a frame with a title bar, a system menu, and a dialog-style border. The modeless dialog box has three auto-radio buttons in it.

```
DLGTEMPLATE modeless1
BEGIN
        DIALOG "Modeless Dialog", 50, 50, 180, 110,
                        FS_TITLEBAR | FS_SYSMENU | FS_DLGBORDER
        BEGIN
                AUTORADIOBUTTON "Retry", 2, 75, 80, 60, 12
                AUTORADIOBUTTON "Abort", 3, 75, 50, 60, 12
                AUTORADIOBUTTON "Ignore", 4, 75, 30, 60, 12
        END
END
```

15.6.1.7.1 Parent/Child/Owner Relationship

The format of the DLGTEMPLATE and WINDOWTEMPLATE resources is very general in order to allow tree-structured relationships within the resource format. The general layout of the templates is this:

```
WINDOWTEMPLATE id
BEGIN
    WINDOW winTop              this is the top level window
    BEGIN
        WINDOW wind1
        WINDOW wind2
        WINDOW wind3
        WINDOW wind4
        BEGIN
                WINDOW wind4
        END
        WINDOW wind5
   END
END
```

In this example, the top level window is identified by winTop. It has 4 child windows, wind1, wind2, wind3, and wind5. wind3 has one child window, wind4. When each of these windows is created, the parent and the owner are set to be the same.

The only time when the parent and owner windows are not the same are when frame controls get automatically create by a frame window.

Note

The WINDOW statements in the example above could also have been a CONTROL or DIALOG statement; they are interchangeable syntactically.

15.6.1.7.2 Pre-defined Control Statements

In addition to the general form of the CONTROL statement, there are special control statements for commonly used controls. These statements define the attributes of the child control windows that appear in the window.

Control statements have the following general form:

```
control-type text, id, x, y, width, height(., style.)
BEGIN
 dialog-statements and/or control-statements and/or window-statements
END
```

Two control statements are exceptions to this general form:

- the EDIT and LISTBOX controls do not have a text field.

The control-type field is one of the keywords described below, defining the type of the control.

text is an ASCII string specifying the text to be displayed. The string must be enclosed in double quotation marks. The manner in which the text is displayed depends on the particular control, as detailed below.

id is a unique integer number identifying the control.

x and y are integer numbers specifying the x and y coordinates of the lower left corner of the control, in dialog coordinates. The coordinates are relative to the origin of the dialog box.

width and height are integer numbers specifying the width and height of the control. The width units are 1/4 the width of a character; the height units are 1/8 the height of a character.

The x, y, width, and height fields can use addition and subtraction operators (+ and -) for relative positioning. For example, 15 + 6 can be used for the x field.

The optional style field consists of one or more of the control styles given later in this chapter in Table 1.2 and the window styles defined in Chapter 2. Styles can be combined using the bitwise OR operator.

The control-type keywords are described below, and their class and default style are given. See Tables 1.1 and 1.2 for a full description of control classes and styles.

FRAME

Description
: Frame control. The style bits of a frame window define which additional frame control windows will be created and initialized when the frame itself is created. Frame style bits are defined in table 1.3. Note that if the text field of this control is non-empty, then a WC_TITLEBAR window will be created even if the FS_TITLEBAR style bit is not included (see below).

  Frame controls created automatically by a frame window will be given default styles and id numbers depending on their class. For example, a WC_TITLEBAR window will be automatically given the id FID_TITLEBAR.

Class Frame

Default Style
: None

LTEXT

Description
: Left-justified text control. A simple rectangle displaying the given text left-justified in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

Class Static

Default Style
: WS_GROUP

RTEXT

Description
Right-justified text control. A simple rectangle displaying the given text right-justified in the rectangle. The text is formatted before it is displayed. Words that would extend past the end of a line are automatically wrapped to the beginning of the next line.

Class Static

Default Style
WS_GROUP

CHECKBOX

Description
A small rectangle (check box) that is highlighted when clicked. The given text is displayed just to the right of the check box. The control highlights the square when the user clicks the mouse in it, and removes the highlight on the next click.

Class Button

Default Style
BS_CHECKBOX, WS_TABSTOP

PUSHBUTTON

Description
A rectangle containing the given text. The control sends a message to its parent whenever the user clicks the mouse inside the rectangle.

Class Button

Default Style
BS_PUSHBUTTON, WS_TABSTOP

LISTBOX

Description
A rectangle containing a list of strings (such as filenames) from which the user can make selections. The LISTBOX control statement does not contain a text field, so the form of the LISTBOX statement is:

```
LISTBOX id, x, y, cx, cy (., style.)
```

The fields have the same meaning as in the other control statements.

Class List box

Default Style
FS_BORDER, FS_VERTSCROLL

GROUPBOX

Description
A rectangle that groups other controls together. The controls are grouped by drawing a border around them and displaying the given text in the upper left corner.

Class Button

Default Style
SS_GROUPBOX, WS_TABSTOP

DEFPUSHBUTTON

Description
A small rectangle with an emboldened outline that represents the default response for the user. The text is displayed inside the button. The control highlights the button in the usual way when the user clicks the mouse in it and sends a message to its parent window.

Class Button

Default Style
BS_DEFPUSHBUTTON, WS_TABSTOP

RADIOBUTTON

Description
A small rectangle that has the given text displayed just to its right. The control highlights the square when the user clicks the mouse in it and sends a message to its parent window. The control removes the highlight and sends a message on the next click.

Class Button

Default Style
BS_RADIOBUTTON, WS_TABSTOP

AUTORADIOBUTTON

Description
Similar to a normal radio button in appearance, but automatically checks itself when clicked. It also unchecks any other AUTORADIOBUTTONs in the same group.

Class Button

Default Style
BS_AUTORADIOBUTTON, WS_TABSTOP

EDIT

Description
A rectangle in which the use can enter and edit text. The control displays a pointer when the user clicks the mouse in it. The user can then use the keyboard to enter text or edit the existing text. Editing keys include the backspace and delete keys. The mouse can be used to select the character or characters to be deleted, or select the place to insert new characters.

The EDIT control statement does not contain a text field, so its form is:

```
EDIT id, x, y, width, height (., style.)
```

The fields have the same meaning as in the other control statements.

Class Edit

Default Style
WS_TABSTOP, ES_LEFT

ICON

Description
An icon displayed in the dialog box. The given text is the name of an icon (not a filename) defined elsewhere in the resource file.

For the ICON statement, the width and height parameters are ignored; the icon automatically sizes itself.

Class Static

Default Style
SS_ICON

15.6.2 Control Classes

Class Meaning

WC_BUTTON
A button control is a small rectangular child window that represents a button that the user can turn on or off by clicking on it with the mouse. Button controls can be used alone or in groups, and can either be labelled or appear without text. Button controls typically change appearance when the user clicks on them.

WC_ENTRYFIELD
: An edit control is a rectangular child window in which the user can enter text from the keyboard. The user selects the control, and gives it the input focus, by clicking the mouse inside it or tabbing to it. The user can enter text when the control displays a flashing pointer. The mouse can be used to move the pointer and select characters to be replaced, or position the pointer for inserting characters. The backspace key can be used to delete characters.

WC_STATIC
: Static controls are simple text fields, boxes, and rectangles that can be used to label, box, or separate other controls. Static controls take no input and provide no output.

WC_LISTBOX
: List box controls consist of a list of character strings. The control is used whenever an application needs to present a list of names, such as filenames, that the user can view and select. The user can select a string by pointing the mouse to the string and clicking a mouse button. Selected strings are highlighted and a notification message is passed to the parent window. A scroll bar can be used with a list box control to scroll lists too long or wide for the control window.

WC_SCROLLBAR
: A scroll bar control is a rectangle containing a thumb and direction arrows at both ends. The scrolling bar sends a notification message to its parent whenever the user clicks the mouse in the control. The parent is responsible for updating the thumb position, if necessary. Scroll bar controls can be positioned anywhere in a window and used whenever needed to provide scrolling input for a window.

Note: A control class name can be used as the class name parameter to the CreateWindow function to create a child window having the control class attributes.

15.6.3 Control Styles

WC_BUTTON Class

Style Meaning

BS_PUSHBUTTON
: Same as PUSHBUTTON statement.

BS_DEFPUSHBUTTON
: Same as DEFPUSHBUTTON statement.

BS_CHECKBOX
: Same as CHECKBOX statement.

BS_AUTOCHECKBOX
: Button automatically toggles its state whenever the user clicks on it.

BS_RADIOBUTTON
: Same as RADIOBUTTON statement.

BS_AUTORADIOBUTTON
: Same as RADIOBUTTON, by automatically checks itself when clicked, and unchecks any other auto-radio buttons in the same group.

BS_3STATE
: Identical to BS_CHECKBOX except that a button can be grayed as well as checked or unchecked. The grayed state is typically used to show that a check box has been disabled.

BS_AUTO3STATE
: Identical to BS_3STATE except that the button automatically toggles its state when the user clicks on it.

BS_GROUPBOX
: Same as GROUPBOX statement.

BS_USERBUTTON
: User-defined button. Parent is notified when the button is clicked. Notification includes a request to paint, invert, and disable the button when necessary.

WC_ENTRYFIELD Class

Style Meaning

ES_LEFT
: Left-justified text.

ES_CENTER
: Centered text.

ES_RIGHT
: Right-justified text.

WC_STATIC Class

Style Meaning

SS_ICON
: Same as ICON control

SS_FGNDRECT
: Foreground color filled rectangle

SS_HALFTONERECT
Halftone filled rectangle

SS_BKGNDRECT
Background color filled rectangle

SS_FGNDFRAME
Box with foreground color frame

SS_HALFTONEFRAME
Box with halftone frame

SS_BKGNDFRAME
Box with Background color frame

WC_LISTBOX Class

Style Meaning

LS_MULTIPLESEL
The string selection is toggled each time the user clicks or double clicks on the string. Any number of strings can be selected.

LS_OWNERDRAW
The list box display is not updated when changes are made.

WC_SCROLLBAR Class

Style Meaning

SBS_VERT
Vertical scroll bar. The scroll bar has the height, width, and position given in the control statement or the CreateWindow call.

SBS_HORZ
Horizontal scroll bar. The scroll bar has the height, width, and position given in the control statement or the CreateWindow call.

All Classes

WS_GROUP
Specifies the first control of a group of controls in which the user can move from one control to the next by using the pointer keys. All controls defined after the first control with WS_GROUP style belong to the same group. The next control with WS_GROUP style ends the first group and starts the next group (i.e., one group ends where the next begins).

WS_TABSTOP
: Specifies one of any number of controls through which the user can move by tabbing. The TAB key moves the user to the next control with WS_TABSTOP style.

15.6.4 Frame Styles

Style	Meaning
FS_TITLEBAR	Title bar
FS_SYSMENU	System menu
FS_MENU	Application menu
FS_MINMAX	Minimize/Maximize box
FS_VERTSCROLL	Vertical scroll bar
FS_HORZSCROLL	Horizontal scroll bar
FS_SIZEBORDER	Wide sizing borders
FS_SIZEBOX	Size box at lower right corner
FS_DLGBORDER	The frame window is created with the FS_DLGBORDER style.
FS_BORDER	Frame window is created with FS_BORDER style
FS_STANDARD	Equal to (FS_TITLEBAR ¦ FS_SYSMENU ¦ FS_MINMAX ¦ FS_WIDESIZE)

15.6.4.1 Directives

The resource directives are special statements that define actions to perform on the script file before it is compiled. The directives can assign values to names, include the contents of files, and control compilation of the script file.

The resource directives are identical to the directives used in the C programming language. They are fully defined in (.CFUN.).

#include filename

Purpose This directive copies the contents of the file specified by filename into your resource script before rc processes the script.

Parameters

filename is an ASCII string, enclosed in double quotation marks, specifying the MS OS/2 filename of the file to be included. A full pathname must be given if the file is not in the current directory or in the directory specified by the INCLUDE environment variable.

The filename parameter is handled as a C string, and two backslashes must be given wherever one is expected in the pathname (for example, root\\sub.) Or, a single forward slash (/) can be used instead of double backslashes (for example, root/sub.)

Example:

```
#include "wincalls.h"
PenSelect MENU
BEGIN
     MENUITEM "black pen", BLACK_PEN
END
```

#define name value

Purpose This directive assigns the given value to name. All subsequent occurrences of name are replaced by the value.

Parameters

name is any combination of letters, digits, or punctuation.

value is any integer number, character string, or line of text.

Examples:

```
#define     nonzero         1
#define     USERCLASS    "MyControlClass"
```

#undef name

Purpose This directive removes the current definition of name. All subsequent occurrences of name are processed without replacement.

Parameters
name is any combination of letters, digits, or punctuation.

Examples:

```
#undef      nonzero
#undef      USERCLASS
```

#ifdef name

Purpose This directive carries out conditional compilation of the resource file by checking the specified name. If the name has been defined using a #define directive, #ifdef directs the resource compiler to continue with the statement immediately after it. If name has not been defined, #ifdef directs the compiler to skip all statements up to the next #endif directive.

Parameters
name is the name to be checked by the directive.

Example:

```
#ifdef Debug
BITMAP errbox errbox.bmp
#endif
```

#ifndef name

Purpose This directive carries out conditional compilation of the resource file by checking the specified name. If the name has not been defined or if its definition has been removed using the #undef directive, #ifndef directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after after the #endif. If name is defined, #ifndef directs the compiler to skip to the next #endif, #else, or #elif directive.

Parameters
name is the name to be checked by the directive.

Example:

```
#ifndef Optimize
BITMAP errbox errbox.bmp
#endif
```

#if constant-expression

Purpose This directive carries out conditional compilation of the resource file by checking the specified constant-expression. If the constant-expression is nonzero #if directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after after the #endif. If constant-expression is zero, #if directs the compiler to skip to the next #endif, #else, or #elif directive.

Parameters

constant-expression is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example:

```
#if Version<3
BITMAP errbox errbox.bmp
#endif
```

#elif constant-expression

Purpose This directive marks an optional clause of a conditional compilation block defined by an #ifdef, #ifndef, or #if directive. The directive carries out conditional compilation of the resource file by checking the specified constant-expression. If the constant-expression is nonzero #elif directs the resource compiler to continue processing statements up to the next #endif, #else, or #elif directive, then skip to the statement after the #endif. If constant-expression is zero, #elif directs the compiler to skip to the next #endif, #else, or #elif directive. Any number of #elif directives can be used in a conditional block.

Parameters

constant-expression is a defined name, an integer constant, or an expression consisting of names, integers, and arithmetic and relational operators.

Example:

```
#if Version<3
BITMAP errbox errbox.bmp
#elif Version<7
BITMAP errbox userbox.bmp
#endif
```

#else

Purpose This directive marks an optional clause of a conditional compilation block defined by an #ifdef, #ifndef, or #if directive. The #else directive must be the last directive before #endif.

Parameters
None.

Example:

```
#ifdef Debug
BITMAP errbox errbox.bmp
#else
BITMAP errbox errbox.bmp
#endif
```

#endif

Purpose This directive marks the end of a conditional compilation block defined by an #ifdef directive. One #endif is required for each #ifdef directive.

Parameters
None.

Chapter 16
Device Drivers

16.1 Device Driver Interface

16.1.1 Overview

A Presentation Manager device driver is used to direct output to devices such as displays and printers, and also to queues. It consists of dynalink code, which runs with IOPL. It should not be confused with an MS OS/2 device driver, which is a specialised piece of code for accessing a general device, including handling interrupts.

The device driver interface resembles the interface to the graphics engine very closely. This gives device drivers the ability to take over many functions of the graphics engine. The device driver entry points correspond exactly to the major function entry points of the graphics engine. The same function numbers are used. The parameters passed to the device driver are exactly the same as those passed to the graphics engine function handlers.

Because the two interfaces are so similar, a caller to a function does not need to know whether the function will be handled by the graphics engine or by the device driver directly. Graphics engine calls are dispatched through a dispatch table. The entries in the dispatch table are far pointers to "Major Function Handlers". Some of these pointers point to the graphics engine, but others point directly to the device driver. On installation the device driver inserts its own pointers into the dispatch table for the functions it wants to handle.

16.1.2 Entry Points

16.1.2.1 Primary Exported Entry Points

All device drivers must export the following entry points:

- Enable
- Disable

Enable is called when the device driver is loaded. It handles driver initialization and construction of the driver's dispatch table.

Display drivers also must export *Cursor* entry points. These are called to update the position and shape of the cursor (or mouse pointer) on the display of the cursor functions.

16.1.2.2 Major Handler Entry Points

As part of its Enable function the device driver installs the addresses of its major function handlers into its logical device dispatch table, the lDispatchTable. The engine will call the major function handlers through the lDispatchTable. The major function handlers handle the dispatching of the required minor functions.

The major function handlers are as follows:

OutputArc
: Major handler 00. Dispatches all minor functions that draw arcs.

OutputLine
: Major handler 01. Dispatches all minor functions that draw lines.

  All device drivers are required to provide this major handler.

OutputMarker
: Major handler 02. Dispatches minor functions that draw markers.

OutputScan
: Major handler 03. Dispatches minor functions that deal with scan lines.

  All device drivers are required to provide this major handler.

OutputFill
: Major handler 04. Dispatches some minor functions that do area filling.

Bitmap
: Major handler 05. Dispatches all minor functions that deal with bitmaps.

  All device drivers are required to provide this major handler.

Textout
: Major handler 09. Dispatches minor functions that display text.

  All device drivers are required to provide this major handler.

Area
: Major handler 0A. Dispatches minor functions for the area accumulation calls.

Bounds
: Major handler 0B. Dispatches minor functions that set and retrieve bound and correlate data.

  All device drivers are required to provide this major handler.

Clip
: Major handler 0C. Dispatches minor functions that deal with the clip region.

Region
: Major handler 0D. Dispatches minor functions that deal with regions.

Transform
: Major handler 0E. Dispatches minor functions that set, retrieve, and calculate transforms.

Attributes
: Major handler 0F. Dispatches minor functions that set and retrieve attributes.

  All device drivers are required to provide this major handler.

Color
: Major handler 10. Dispatches minor functions that deal with color tables.

  All device drivers are required to provide this major handler.

Query
: Major handler 11. Dispatches minor functions that query device capabilities and parameters and also the escape function for direct access of driver function.

  All device drivers are required to provide this major handler.

16.1.3 Function Parameters

In the function descriptions, each minor function definition gives the name of the function, followed by the syntax of the parameters to the function. The function to be performed is described, followed by a more detailed explanation of the function specific parameters. For all functions, the parameters *u32_FuncNo* (Function Number), and *u32_DcH* (Device Context Handle) are as described below, unless otherwise stated.

16.1.3.1 Stack Arguments

Each call to a major function handler passes a number of parameters which depend upon the particular minor function. The last parameter pushed is the function number, from which the particular minor function, and therefore the number and meaning of the other parameters, can be determined.

PASCAL calling conventions are used in making the call. The device driver is responsible for setting up its own stack frame, and for removing all arguments when it returns.

16.1.3.2 Function Number

The format of the function number parameter is as follows:

minor_function (Bits 0-7)
: The minor function number within the particular major function.

major_function (Bits 8-15)
The major function number.

command_flags (Bits 16-31)
Flags which identify the operations which are to be performed, and give other information. See below.

16.1.3.3 Command Flags

The meaning of the command flags parameter is as follows (the bit numbering views *command_flags* as a 16-bit value, with bit 0 the low order bit).

If more than one of the bits is set then more than one operation needs to be performed. If none of the bits are set, there may still be something that needs doing. An example is that the current position must be updated on a drawing command even if the COM_DRAW bit is not set.

COM_DRAW (Bit 0)
Draw the figure.

When COM_DRAW is set, the device driver must actually draw the requested figure on the device or bitmap. If the bit is off, then any functions that would normally be performed in addition to drawing must still be done (like updating the current position).

COM_BOUND (Bit 1)
Calculate the bounding rectangle.

When COM_BOUND is set, the device driver must calculate the bounding rectangle for the given figure. The engine should then be called to accumulate the resulting rectangle. Use the AccumulateBounds call.

All device drivers must be able to calculate bounds on any figure they can draw.

COM_CORR (Bit 2)
Calculate correlation with the Pick Window.

When COM_CORR is set, the device driver must determine whether the given figure intersects the Pick Window that was set by SetPickWindow. If an intersection is detected, the driver should return the appropriate return code for correlation, as indicated in the function description.

Only display device drivers are required to calculate correlations.

COM_ALT_BOUND (Bit 3)
Equivalent to COM_BOUND. The device driver should calculate the bounding rectangle if either this bit or COM_BOUND is set (or both).

(The above functions only apply to drawing functions and should be ignored for other functions.)

COM_TRANSFORM (Bit 5)
Coordinates are to be transformed.

When COM_TRANSFORM is set, any coordinates given are not yet transformed to device coordinates. The Convert function must be used to transform them. If the bit is clear, then a simulation has already done the conversion, and the device driver can assume that all coordinates are device coordinates.

All other bits should be ignored.

16.1.3.4 Device Context Handle

This parameter is the handle of the Device Context. See the description of GetDCPointer for converting this to an actual pointer.

The device driver is given access to a ULONG (4 bytes) of information starting at byte offset 4 in the DC. This ULONG is under total control of the device driver, except that it will be zeroed when the DC is created. It is expected that the device driver will store an index or pointer here that will help it locate its attribute instantiations for this DC. We will refer to this ULONG as the DC Magic Number.

Also, at offset 8 of the DC, the device driver has access to the hLogicalDevice. This is a 32-bit handle for the logical device that the DC belongs to. This allows the device driver to recognize its own DC objects. This lets the device driver do its own error checking, preventing a stopover in the engine layer.

The logical device handle is useful when doing a BitBlt operation from one device bitmap to another. In this case, both bitmaps should be in DC's on the same logical device and the handle allows the driver to check this.

16.1.3.5 Register Arguments

The following registers will have defined values when the device driver major handler is called from the lDispatchTable:

CX:DX = The DC magic number. CX contains the high order word.

ES = The DC Segment.

These assignments are provided as an optimization only. The information provided is obtainable elsewhere. We do not expect that device driver major function dispatchers will be able to use these at all if they are not written in assembler.

The default engine simulations will not depend on these assignments. This means that a device driver major function dispatcher can destroy these values and branch to a default simulation without any problems.

16.1.3.6 Return Values

The device driver must return an error code in DX:AX when it has completed the function call. The return values are described under each function.

16.1.3.7 Register Content Preservation

On completion of each device driver function, DX:AX will contain any error code. Registers BX, CX, and ES may be destroyed. All other registers must be preserved.

16.1.3.8 Stack Usage

The device driver can assume that 500 (decimal) bytes of space are available on the stack when a device driver routine is called. The device driver code can use this space freely, but it must not exceed this limit.

If there is a danger of running out of stack space, the device driver should allocate its own stack space, switching to it on entry, and reverting to the original stack on exit.

16.1.4 Calling Simulations

A device driver and any simulation may call the graphics engine through a lower level interface than GreEntry. This entry point is called SimulationEntry.

SimulationEntry should be called as a FAR call, with exactly the arguments as required for the particular function being called, including the function number. This is accomplished by restoring the original stack frame before making the call.

The Command word may be set to request any combination of COM_DRAW, COM_CORR, and COM_BOUND.

16.1.5 Bitmap Simulations

Simulations are provided for some standard bitmap formats. Devices that use bitmaps in these formats may call on the simulations to do any drawing function, like arcs and polylines for example. This allows device drivers for dot matrix printers and such to share common code for drawing on bitmaps.

The supported bitmap formats are:

Bitcount	Planes
1	1
8	1
24	1
4	1

Note: Device Drivers must be able to translate from all the standard formats to their own internal format. This allows a bitmap created on one device to be displayed on another.

All device drivers are required to support the PolyLine and PolyShortLine calls for drawing on bitmaps. However, to prevent the same code from appearing in many device drivers, a non-display device driver may rely on the PolyLine and PolyShortLine code in the display driver to actually draw on bitmaps. Of course, display device drivers cannot do this, and are required to actually draw on bitmaps themselves.

A typical use of this trick is by dot matrix printers. When a DC is created for the printer, the printer device driver does the following:

1. Call CreateDC to make a memory DC for the "DISPLAY" device.
2. Call CreateBitmap to make a bitmap compatible with the display DC.
3. Select the bitmap into the DC with SelectBitmap.

When a drawing function, like PolyLine, is called for drawing on the printer DC, the printer driver passes the call along by calling PolyLine on the display DC with SimulationEntry. The display driver will set the appropriate bits in the bitmap.

When all drawing commands to the printer DC are completed, as indicated by the EndDoc escape call, the printer driver can retrieve the bits from the bitmap with GetBitmapBits. The printer then prints the bits on the page.

When the printer DC is deleted, the printer device driver should deselect the bitmap from the display DC, delete the bitmap, and delete the display DC.

16.1.6 Journalling

For support of banding printers, drivers need to be able to journal and repeatedly play back the drawing calls they receive. Functions are provided in the engine to perform this. See the section entitled "Journalling" in the Device Drivers Chapter.

16.1.7 Serialization and Locking

The device driver should be designed to be re-entrant. It must assume that it can be called by two or more different threads at any time.

A device driver is always called by the Graphics Engine when the engine is outside its critical sections. This implies that the device driver can afford to take a long time to implement a particular function on a given thread. For example, it IS possible for the device driver to access a resource on disk or to put up a dialog box for additional information.

It is often necessary for a device driver to serialize access to internal resources - the actual hardware, for example. The driver code has access to all the normal serialization mechanisms available to MS OS/2 code running at ring 2:

- CLI/STI
- RAM Semaphores
- System Semaphores

The device driver writer can choose whichever of these is suitable for the particular circumstances. The only caveat is that the device driver should NEVER call another system component during a critical section. This includes the file system, the graphics engine (via SimulationEntry) or the MS OS/2 Presentation Manager API.

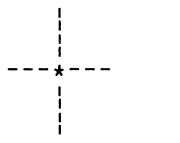
16.1.8 Cursors

All display drivers must support a "cursor" for the pointing device. The cursor is a small graphics image which is allowed to move around the screen independently of all other operations to the screen, and is normally bound to the location of the pointing device. The cursor is non-destructive in nature, i.e., the bits underneath the cursor image are not destroyed by the presence of the cursor image.

A cursor consists of an AND mask and an XOR mask, which give combinations of 0's, 1's, display, or inverse display.

AND	XOR	DISPLAY
0	0	0
0	1	1
1	0	Display
1	1	Not Display

The cursor also has a "hot spot", which is the pixel of the cursor image which is to be aligned with the actual pointing device location.



For a cursor like this, the hot spot would normally be the *, which would be aligned with the pointing device position

The cursor may be moved to any location on the screen or be made invisible. Part of the cursor may actually be off the edge of the screen, and in such a case only the visible portion of the cursor image is displayed.

Logically, the cursor image isn't part of the physical display surface. When a drawing operation coincides with the cursor image, the result is the same as if the cursor image wasn't there. In reality, if the cursor image is part of the display surface it must be removed from memory before the drawing operation may occur, and redrawn at a later time.

This exclusion of the cursor image is the responsibility of the display driver. If the cursor image is part of physical display memory, then all output operations must perform a hit test to determine if the cursor must be removed from display memory, and set a protection rectangle wherein the cursor must not be displayed. The actual cursor image drawing routine must honor this protection rectangle by never drawing the cursor image within its boundary.

The cursor drawing primitives reside in the Ring 2 display driver. These primitives may be called at various times from many different places, so the cursor code must protect itself via a semaphore (any and all protection is the sole responsibility of the display driver). Since cursor drawing can be a time consuming operation, the driver must also protect itself against reentrancy.

The conditions under which the cursor drawing primitives may be called are as follows:

1. One of the following
    1. A mouse movement occurs. Mouse movements are passed to the MoveCursor routine at interrupt time.
    2. The window manager is setting a new cursor position.

The current cursor location must be set to the given coordinates. If the cursor is visible, it will be drawn at the new location. If the cursor is off (a NULL cursor), or if the cursor has been excluded, then no updating of the image is required.

If the cursor is on and the new cursor position will cause the cursor to be excluded, it must be removed from the screen.

In either case, the real cursor position must be updated to the passed (x,y).

Once the cursor has been drawn, a check must be made to see if a new location was given for the cursor, and if it has moved again, be drawn at the new location (or be excluded because it has moved into the protection rectangle). This implies that a real (x,y) and a cursor_shape (x,y) be maintained.

```
void MoveCursor(abs_x,abs_y)
SWORD abs_x;                            // x coordinate of cursor
SWORD abs_y;                            // y coordinate of cursor
{
   USHORT    old_busy;

   enter_crit();                        // Updating the real X,Y is
   real_x = abs_x - hot_x;              //      a critical section
   real_y = abs_y - hot_y;
   old_busy = IS_BUSY;                  // Try for screen semaphore
   swap(screen_busy,old_busy);
   leave_crit();

   if (old_busy == NOT_BUSY)
   {
       while(cursor positions disagree)
       {
             if (cursor hidden || already excluded)
             {
                 screen_busy = NOT_BUSY;
                 return();
             }
             if (newly excluded)
             {
                 cur_flags = CUR_EXCLUDED;
                 cursor_off();
                 screen_busy = NOT_BUSY;
                 return();
             }
             draw_cursor();             // can actually draw cursor
       }
       screen_busy = NOT_BUSY;          // others can have the screen
                                        // now
   }
   return();
}
```

2. A new cursor image is being set. When a new cursor image is set, the old cursor image, if any, must be removed from the screen before the new cursor is set. The hot spot of the old cursor and the new cursor must be aligned. This code must also protect itself from any of the drawing primitives, or from the interrupt thread moving the cursor.

```
void SetCursor(lp_cursor)
CURSOR far *lp_cursor
{
   USHORT    old_busy;

   old_busy = IS_BUSY;                  // Try for screen semaphore
   if (swap(screen_busy, old_busy) == IS_BUSY)
       return();

   disable_interrupts;                  // Treat as a critical section
   cur_flags = CUR_OFF;                 // Assume a null cursor;
   real_x += hot_x;                     // Remove hot spot adjustment
   real_y += hot_y;                     //     from real (X,Y) position
   hot_x = hot_y = 0;                   // Don't want hot spot adjustments
   enable_interrupts;                   // Interrupt can play with real x & y
   cursor_off();                        // Remove old cursor from s
   if (lp_cursor)                       // If there is a new cursor
   {
       copy(cur_cursor,lp_cursor);      // Copy cursor header information
       move_cursors();                  // Move the patterns, adj. hot spot
       disable_interrupts;              // Treat as a critical section
       hot_x = cur_cursor.csHotX        // Save X hot spot adjustment
       hot_y = cur_cursor.csHotY        // Save Y hot spot adjustment
       real_x -= hot_x;                 // Adjust real (X,Y) for the
       real_y -= hot_y;                 //     hot spot
       cur_flags = CUR_EXCLUDED;        // Show excluded, but not hidden
       enable_interrupts;
   }
   screen_busy = NOT_BUSY;              // Others can have the screen now
}
```

3. A timer interrupt occured. Approximately every 1/4 second, the Window Manager will call CheckCursor. This allows a lazy redraw of the cursor whenever it has been removed from the screen. Use of this function is optional.

   If the cursor is currently invisible, and can now become visible, then it should be drawn. If while the cursor was being drawn, it moved, then it must be drawn at the new location. If it moved into the protection rectangle, then it must be taken down again.

   This code must protect itself from any of the drawing primitives, or from the interrupt thread moving the cursor.

```
void CheckCursor();
{
   USHORT    old_busy;

   if (swap(screen_busy,old_busy) == screen_busy)
       return();                        // cannot access the screen

   if (cursor is off || cursor not excluded)
   {                                    // nothing to do
       screen_busy = NOT_BUSY;
       return();
   }

   // The cursor is currently excluded. If it is now unexcluded,
   // it must be drawn.

test_if_unexcluded:
```

```
    enter_crit();
    if (cursor unexcluded)
    {
        leave_crit();
        draw_cursor();                  // draw cursor at new location
        cur_flags = 0;                  // show cursor is on and unexcluded
        enter_crit();
        if (cursor positions disagree)
              goto test_if_unexcluded;  // moved while we were drawing it
        screen_busy = NOT_BUSY;
        leave_crit();
        return();
    }
    leave_crit();

    // Must test to see if the cursor became excluded after we
    // just brought it back.

    if (cursor is excluded)
    {
        cursor_off();
        cur_flags = CUR_EXCLUDED;
    }

    screen_busy = NOT_BUSY;             // others can have the screen now
    return();
}
```

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

16.1.9 Miscellaneous

This document defines the function of the Graphics Engine as used by Presentation Manager. This component provides output graphics, alphanumerics and raster operations to output devices (e.g., screen windows) and to bitmaps.

The following points are worth noting:

- Output graphics primitives (such as lines) are based on the following:
  - — Coordinates to drawing primitives are passed in World Coordinate Space.
  - — The viewing limits are defined in Graphics Model Space Coordinates.
  - — The pick window is defined in Page Coordinates.
  - — Boundary data (bounds) are collected in Graphics Model Space Coordinates. (It is possible to implement bounds collection in Page Coordinates, but then the graphics engine must transform them back to Graphics Model Space Coordinates before returning them to the GPI, and also when the window/viewport transform changes.)

  - — Drawing is repeatable (the same sequence of primitives turns on the same pels) so that dynamic objects can be drawn and removed by using XOR mode.
  - — Clipping to the output region is perfect at the pel level, so that primitives will join up if drawn in neighboring regions.
  - — The line type position is reset by the appropriate orders only.
- Fonts are stored by the graphics engine. Both are referenced by LCID and the engine is responsible for resolving whether an LCID refers to a loaded font, or whether a base set should be used.
- Base line types, patterns and markers (vector and image, resolved by the marker precision) are provided by the engine. The Engine will ensure that base symbol sets/fonts are available for each device. These will include one vector and at least one size of image for each of the available codepages.
- The graphics engine does a certain amount of error checking, for example that the composite transformation does not overflow. In these cases, the error is logged.
- The graphics engine resolves contention for its own internal data structures. All device drivers must resolve contention for their own data structures and hardware registers.
- The origin for all graphics coordinates, is at the bottom left (i.e., y increases upwards).
- To avoid deadlock, no part of the engine or any device driver may call another component inside a critical section.

16.1.10 Syntax Issues in Defining the Engine and Device Driver Interfaces

This section defines the conventions for calls, parameters and datatypes. This definition has been chosen to be unambiguous and easy to code in our respective implementation languages (C and ASSEMBLER).

16.1.10.1 Parameter Conventions

All parameters will be prefixed by one of the following to indicate their data type:

```
Prefix   C type

p32_     *type far
u32_     unsigned long
s32_     long int
```

The u32 prefix will be used for unsigned and non arithmetic variables variables occupying 32 bits.

To avoid problems with alignment on the I80386 microprocessor, all parameters will be 32 bits. Flags must therefore be specified by saying which bit number (or numbers) they occupy, according to the bit numbering scheme listed above.

Here are some examples:

```
u32_MixMode    Mix Mode passed as Unsigned Long
s32_x          X coordinate passed as Long Int
s32_y          Y coordinate passed as Long Int
p32_xy         Long pointer to an xy array (of Long Int)
```

To limit the number of parameters passed, coordinates will not be passed as explicit parameters, instead a pointer to an array of one or more xy coordinate pairs is passed.

16.1.10.2 Calling Conventions

The parameter passing mechanism used will be PASCAL calling convention. As an example consider a hypothetical entry point called ROUTINE(U32_P1, U32_P2, U32_P3). This routine would be called from PASCAL by

```
ROUTINE(U32_P1, U32_P2, U32_P3);
```

or from C by reversing the order of the parameters

```
CALL ROUTINE(U32_P3, U32_P2, U32_P1);
```

both of which generate the following assembler

```
PUSH U32_P1+2
PUSH U32_P1
PUSH U32_P2+2
PUSH U32_P2
PUSH U32_P3+2
PUSH U32_P3
CALL ROUTINE * This is a far call
```

Note: that the called routine must pop the parameters off the stack (done with RETF 12 in assembler, or declaring the entry point in C with the PASCAL calling convention).

The following registers are saved and restored by the Engine around each engine function call: SI, DI, DS and BP. DX:AX will contain any return code. Registers BX, CX, and ES may be destroyed.

16.1.10.3 Definition of Records in C

To avoid alignment problems, records will be constructed with 16 and 32 bit fields. Varying length fields will go at the end. Flags contained in a field in a record will be recorded by the bit number or numbers they occupy. All 32 bit quantities will be aligned on 32 bit boundaries, so that access will be more efficient on the I80386 microprocessor. The 16 bit types allowed in records identified by prefixes

```
u16_     unsigned int
s16_     int
```

Examples of records are

```
RECORD record1,
  field1 s32,  /* Description of Long Int field1     */
  field3 p32,  /* Description of Long Pointer field3 */
  field2 u16;  /* Description of Unsigned Int field2 */
               /* Bit 0 flag means ... */
               /* Bit 1 flag means ... */
               /* And so on                          */
```

Arrays are specified by putting the dimension(s) after the names of the fields.

The first element in an array is addressed with index 0.

Contiguous elements of an array in a structure are assumed to be contiguous in storage. Contiguous elements of an array of structures are assumed to be contiguous in storage.

```
RECORD record2(100), /* Array of 0..99               */
  field1 s32,        /* Description of field1        */
  field2(20) s32,    /* Array of 0..19               */
  field3 p32;        /* Description of field3        */
/* There are 100 copies of field1 and field3, 2000 of field2 */
```

16.1.10.4 Handles

All handles and pointers are 32 bit.

16.1.10.5 Coordinates

All coordinates are signed 32 bit local (Intel) format at the engine interface unless otherwise stated. These coordinate values are restricted to lie within the range 'FFFF8000'X - '00007FFF'X. A pointer to the coordinate pair(s) is passed in each case.

16.1.10.6 Bit Definitions

All references to individual bits in this document follow the Intel format i.e.,

for 16 bit numbers:

```
7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8
              LSB MSB
```

for 32 bit numbers:

```
7 6 . . . 1 0 15 14 . . . 9 8 23 22 . . . 17 16 31 30 . . . 25 24
            LSB                                 MSB
```

where LSB is the least significant bit and MSB is the most significant bit.

16.1.11 The Dispatch Table

16.1.11.1 Device Driver Installation

When the device driver module is loaded, the graphics engine will call the Enable function.

Among the arguments to this function will be a pointer to the lDevice-DispatchTable. This table will already be filled with the addresses of the graphics engine default major handlers. (Copied from the Default-DispatchTable.) The device driver must overwrite the entries in this table that correspond to the functions it wishes to handle.

16.1.11.2 Device Driver Function Handling

The device driver should execute a FAR return from an optional function only when it completes all processing required for that function. If it cannot complete the function, it must pass control to the engine default handler for that major function. The address of the default major handler can be found in the DefaultDispatchTable, which is a globally readable object.

A driver may not be able to complete processing in cases when it cannot handle certain combinations of attributes, like wide styled curves, for example.

Any minor function number that the driver does not recognize must be passed to the default major handler. This will allow device drivers to continue to operate even if the interface is expanded.

Because the interface may be expanded to include functions that even the most complete device driver cannot know about, the engine default handler must be allowed access to any functions that modify drawing attributes. The device driver should record the new attributes and perform any work required for their instantiation, and then pass the call to the default major handler for that function. The attribute is recorded twice, but the engine is capable of taking over drawing at any time. The calls that must be shared in this way are:

- SetArcParameters
- SetCurrentPosition

16.1.11.3 Simulation Installation

Simulations are installed during system initialization. Installation of simulations differ in that the desired functions are placed directly in the DefaultDispatchTable. In this way, they replace the default engine major handlers, but are not distinguishable from them.

Simulations should make a local copy of the pointers they are replacing. This will allow them to use the engine handlers if they are ever needed.

16.1.12 Primary Function Definitions

16.1.12.1 Control Functions

Short list of the functions:

- ErasePS
- SetProcessControl
- GetProcessControl
- ResetBounds
- SetPickWindow
- GetPickWindow
- GetBoundsData
- QueryCharCorr
- Death
- Resurrection
- LockDevice

- UnLockDevice
- SetCursor
- DeviceSetCursor

ErasePS(u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

Erases the output media associated with the specified Device Context handle to the global standard default background color, on devices that are capable of supporting this operation (i.e., no operation is performed for printers or plotters).

This operation is unaffected by the draw process control bit and is unaffected by any application defined clipping.

Returns: *BOOL*

0 error
1 ok

SetProcessControl(u32_Mask, u32_ProcessFlags, u32_DcH, u32_FuncNo)

Provides a means of turning draw, boundary computation and correlation on and off.

Bounds are returned in graphics model space coordinates. If a composite transform is applied to the drawing primitives, the bounds values need to be back transformed before merging with the previous bounds values. Bounds computation is performed on unclipped primitives.

Correlation is performed in Page Coordinate Space on the output of primitives that have been clipped to the Viewing Limits and Graphics Field only. *Note:* Boundary determination and correlation are performed for all the functions specified in this document that draw to the output media, excluding the alphanumerics functions.

Parameters:

u32_Mask
: Only process flags with the corresponding u32_Mask bit set to 1 are modified.

u32_ProcessFlags
: Contains the following flags:

Bit 0 draw
Set to *'1'B* to indicate that drawing primitives should appear on the screen.

If this flag is off then, except for ErasePS, no output operations are displayed (i.e., BitBlt, PaintRegion, SetPel, drawing primitives, etc., are not displayed).

This flag has no effect for ErasePS.

Bit 1 bounds
Set to *'1'B* to indicate that bounds should be collected.

Bit 2 correlate
Set to *'1'B* to indicate that correlation should be performed.

Bit 3 userbounds
Set to *'1'B* to indicate that bounds should be collected for the user interface.

Returns: *BOOL*

0 error
1 ok

GetProcessControl(u32_DcH, u32_FuncNo)

Returns the process control flags:

Bit 0 returns the draw process flag.

Bit 1 returns the bounds process flag.

Bit 2 returns the correlate process flag.

Bit 3 returns the user bounds process flag.

Bit 4 set to '1'B to indicate that an area definition is in progress.

Bit 5 set to '1'B to indicate that a path definition is in progress.

Bits 6-31
reserved.

Returns: *long int*

-1 error
>=0 process flags

ResetBounds(u32_Flags, u32_DcH, u32_FuncNo)

This resets the bounds values to their initial default values.

Parameters:

u32_Flags
Contains the following flags:

Bit 0 GpiBounds
Set to *'1'B* to indicate that the Gpi interface bounds rectangle is to be reset.

Bit 1 UserBounds
Set to *'1'B* to indicate that the User interface bounds rectangle is to be reset.

Returns: *BOOL*

0 error
1 ok

SetPickWindow(p32_PickWindow, u32_DcH, u32_FuncNo)

```
place=inline  frame=box

struct ARGUMENTS {
        ULONG    u32_FuncNo;
        ULONG    u32_DcH;
        ULONG*   P32_PickWindow;
};
```

This sets the position and size of the pick window, in page coordinate space, for subsequent correlation operations.

The boundary of the pick window is included in the area correlated upon.

Parameters:

p32_PickWindow
Points to an array containing the minimum and maximum xy coordinate pairs of the window:

(s32_xmin, s32_ymin, s32_xmax, s32_ymax)

The data in the array may be overwritten.

Returns: *BOOL*

0 error
1 ok

GetPickWindow(p32_PickWindow, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG    u32_FuncNo;
        ULONG    u32_DcH;
        ULONG*   P32_PickWindow;
};
```

This returns the position and size of the pick window, in page coordinate space.

Parameters:

p32_PickWindow
: The address at which to return an array containing the minimum and maximum xy coordinate pairs of the window:

  (s32_xmin, s32_ymin, s32_xmax, s32_ymax)

Returns: *BOOL*

0 error
1 ok

GetBoundsData(u32_Style, p32_BoundsData, u32_DcH, u32_FuncNo)

This returns the bounding rectangle of previous drawing primitives in graphics model space coordinates for Gpi bounds and device coordinates for User bounds.

Parameters:

u32_Style
: Valid values are:

  0 — Indicates that the Gpi bounds rectangle is to be returned.

  1 — Indicates that the User bounds rectangle is to be returned.

p32_BoundsData
: The address at which to return the bounds data as an array of two xy pairs, such that:

  s32_xmin
  : specifies the minimum x bound value.

  s32_ymin
  : specifies the minimum y bound value.

  s32_xmax
  : specifies the maximum x bound value.

  s32_ymax
  : specifies the maximum y bound value.

  Null bounds data is returned by setting the x and y bounds to their initial default values.

Returns: *BOOL*

0 error
1 ok

QueryCharCorr (u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This function returns an offset indicating which character within a character string was selected, the last time a character string primitive returned a successful correlation hit. If more than one character in the string was selected, the offset of the first is returned.

A value of zero indicates the first character in the string. A negative value indicates that no string has been correlated on.

Returns: *long int*

```
-4 error
-1 no offset
>=0 character offset
```

Death (u32_DcH, u32_FuncNo)

This function is used to provide notification of a screen group switch into another screen group.

This function goes directly to the DDI.

Returns: *BOOL*

```
0 error
1 ok
```

Resurrection (u32_DcH, u32_FuncNo)

This function is used to provide notification of a screen group switch back from another screen group.

This function goes directly to the DDI.

Returns: *BOOL*

```
0 error
1 ok
```

LockDevice(u32_DcH, u32_FuncNo)

```
place=inline  frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This function is used to synchronize use and update of VisRegion. It allows all current and pending drawing to complete and blocks any further requests by other threads to draw.

Upon exit, the only thread that is allowed to continue screen I/O is the one that acquires the lock.

All screen I/O operations by other threads will be blocked until UnlockDevice is called.

LockDevice and UnlockDevice are used mainly in the critical sections of visible region calculations.

To prevent deadlock, it is guaranteed that no Death or Resurrection will be called by the window manager while the VisRegion is locked.

Returns: *BOOL*

0 error
1 ok

UnLockDevice(u32_DcH, u32_FuncNo)

```
place=inline  frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This function is used to synchronize use and update of VisRegion. It allows all pending screen I/O operations blocked by LockDevice to continue.

LockDevice and UnlockDevice are used mainly in the critical sections of visible region calculations.

Returns: *BOOL*

0 error
1 ok

SetCursor(p32_xy, u32_CursorBmapH, u32_DcH, u32_FuncNo)

Sets the cursor bitmap that defines the cursor shape. Each call replaces the previous bitmap with that identified by u32_CursorBmapH. If this is null, the cursor has no shape and its image is removed from the display screen.

Parameters:

p32_xy A far pointer to *u32_xHotSpot, u32_yHotSpot* which define the x,y position within the cursor of the "hot spot."

u32_CursorBmapH
The bitmap handle to be used for the cursor image.

Returns: *BOOL*

0 error
1 ok

DeviceSetCursor (p32_xy, u32_BmapH, u32_DcH, u32_FuncNo)

```
place=inline  frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_BmapH;
        ULONG*  p32_xy;
};
```

Sets the cursor bitmap that defines the cursor shape. Each call replaces the bitmap with that pointed to by u32_BmapH. If u32_BmapH is null, the cursor has no shape and its image is removed from the display screen.

Parameters:

p32_xy A far pointer to *u32_xHotSpot, u32_yHotSpot* which define the x,y position within the cursor of the "hot spot."

u32_CursorBmapH
The bitmap handle to be used for the cursor image.

Returns: *BOOL*

0 error
1 ok

16.1.12.2 Attribute Functions

The following functions pass individual set attribute orders to the Graphics Engine.

Short list of the functions:

- SetArcParameters
- SetCurrentPosition
- SetPatternOrigin
- GetArcParameters
- GetCurrentPosition
- GetPatternOrigin
- SetLineTypeGeom
- QueryLineTypeGeom
- SetStyleRatio

SetArcParameters(p32_AttributeData, u32_DcH, u32_FuncNo)

```
place=inline frame=box
struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_ArcParameters;
};
```

Sets the arc parameters to the specified values.

Parameters:

p32_AttributeData
: Points to a 4 element array containing the integer values for the arc parameters:

  (s32_p, s32_q, s32_r, s32_s)

The arc parameters define the shape and orientation of an ellipse which is used for subsequent Arc, FullArc, and PartialArc functions. For all of these functions except Arc, they also determine the direction of drawing, as follows:

```
s32_p * s32_q > s32_r * s32_s   anticlockwise
s32_p * s32_q < s32_r * s32_s   clockwise
s32_p * s32_q = s32_r * s32_s   straight line
```

Also except for Arc, they define the nominal size of the ellipse, although this may be changed by using the multiplier. For Arc, the size of the ellipse is determined by the three points specified on Arc.

The arc parameters define a transformation that maps the *unit circle* to the required ellipse, placed at the origin (0,0):

```
x' = p.x + r.y
y' = s.x + q.y
```

If p.r + q.s = 0, then the transform is termed orthogonal, and the line from the origin (0,0) to the point (p,s) is either the radius of the circle, or half the major axis of the ellipse. The line from the origin to the point (r,q) is either the radius of the circle, or half the minor axis of the ellipse.

For maximum accuracy orthogonal transforms should be used.

The standard default values of arc parameters (which define a unit circle) are

```
p = 1    r = 0
s = 0    q = 1
```

The arc parameters transformation takes place in World Coordinates. Any other non-square transformations in force will change the shape of the figure accordingly.

Returns: *BOOL*

0 error
1 ok

SetCurrentPosition(p32_xy ,u32_DcH, u32_FuncNo)

```
place=inline frame=box
struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_xy;
};
```

Sets the current x,y position to the specified value and resets the line type sequence. If the current context is 'in area', then a figure closure line is generated if necessary, this may cause a correlation hit on an area boundary to occur. The current position should only be correlated on, and or be merged into the bounds, if it is actually used in a drawing primitive. Thus, for example, the sequence SetCurrentPosition to P1, SetCurrentPosition to P2, Polyline to P3, will not merge P1 into the bounds or correlate on it.

Parameters:

p32_xy Points to the integer values *(s32_x, s32_y)* of the new current position in world coordinate space.

Returns: *BOOL*

0 error
1 ok

SetPatternOrigin(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_xy;
};
```

Sets the pattern reference point used for blting and filling to the specified value.

Parameters:

p32_xy Points to *(s32_x, s32_y)*, the origin of the pattern relative to the origin (or window on the screen) in world coordinates.

The pattern reference point is the point which the origin of the area filling pattern maps to. The pattern is mapped into the area

to be filled by conceptually replicating the pattern definition in horizontal and vertical directions.

Since the pattern reference point is subject to all of the transforms, if an area is moved by changing a transform and redrawing, the fill pattern will also appear 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. The pattern reference point is not subject to clipping, although of course the area to be filled will be.

The pattern reference point applies to filled areas and to FloodFill.

Returns: *BOOL*

0 error
1 ok

GetArcParameters(p32_AttributeData, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_ArcParameters;
};
```

Returns the current arc parameters p, q, r, s in a 4 element array:

(s32_p, s32_q, s32_r, s32_s)

Parameters:

p32_AttributeData
: Specifies the return address for the data.

Returns: *BOOL*

0 error
1 ok

GetCurrentPosition(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_xy;
};
```

Returns the current position as an x,y coordinate pair:

(s32_x, s32_y)

Parameters:

p32_xy Specifies the return address for the data.

Returns: *BOOL*

0 error
1 ok

GetPatternOrigin(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_xy;
};
```

Gets the origin of the pattern used for blting and filling.

Parameters:

p32_xy Points to the return address for *(s32_x, s32_y)*, the origin of the pattern relative to the origin (or window on the screen) in world coordinates.

Returns: *BOOL*

0 error
1 ok

SetLineTypeGeom(u32_options, u32_count, p32_lengths, u32_DcH, u32_FuncNo)

Note: SetLineTypeGeom is not required for the first release of Presentation Manager.

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_lengths;
        ULONG u32_count;
        ULONG u32_options;
};
```

Sets the geometric line-type attribute to the specified values. This is the line-type which will be used if geometric thick lines are being drawn (see line_geometric_width).

A non-solid geometric line-type consists of a sequence of 'on' and 'off' runs which gives the appearance of a dotted, dashed, etc., line. The lengths of the runs are specified in World Coordinates, so that they are subject to all of the transforms, in the same way that geometric line thickness is.

The system maintains position within the line-type definition, so that, for example, a curve may be implemented as a polyline. However, certain functions cause position to be reset to the start of the definition. These are:

- SetLineTypeGeom
- SetCurrentPosition
- SetModelTransform
- SetWindowViewportTransform
- SetPageWindow
- SetPageViewport

Parameters:

u32_options
: Option flags. This consists of 32 flags (with 0 the least significant). These may be used in combination. Each set bit has the following meaning:

  LWG_INIT (bit 0)
  : If set, the first run is 'on'. Otherwise, it is 'off'.

  LWG_REP (bit 1)
  : If set, runs repeat from the second value. Otherwise, they repeat from the first. In either case, the value of LWG_INIT is ignored for repeats.

u32_count
: The number of elements in the array pointed at by p32_lengths.

p32_lengths
: A far pointer to an array, containing *u32_count* elements, which specifies the run lengths, in world coordinates. Each array element is of type s32 (long).

Returns: *BOOL*

0 Error
1 OK

QueryLineTypeGeom(p32_options, u32_count, p32_lengths, u32_DcH, u32_FuncNo)

Note: QueryLineTypeGeom is not required for the first release of Presentation Manager.

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_lengths;
        ULONG u32_count;
        ULONG* p32_options;
};
```

Returns the geometric line-type attribute.

Parameters:

p32_options
: A far pointer to a variable in which the option flags are returned. See SetLineTypeGeom.

u32_count
: Set by the application to the number of elements in the array pointed to by *p32_lengths.*

p32_lengths
: A far pointer to an array containing *u32_count* elements, in which the run lengths, in world co-ordinates, are returned.

Returns: *long int*

```
-1 Error
>=0 Count of number of elements returned
```

SetStyleRatio(p32_ratio, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG* p32_Ratio;
};
```

Specifies the ratios to be used when drawing styled lines.

This function is used for banding printers which use display DDs to write into bitmaps. When drawing a styled line, equal length dashes (and dots) must be maintained in all directions. Printer driver calls will be redispatched to the display driver for banding in printers, and must be able to set this aspect ratio so that the printer can have the display driver draw correct lines.

```
Sample ratios
        5,12,13     - cga
        10,10,14    - all one to one devices
```

Parameters:

p32_Ratio

Points to three 16:16 fixed point numbers. These define the sides of a right angle triangle, which corresponds to the aspect ratio of the pels a line is drawn on.

Returns: *BOOL*

0 error
1 ok

16.1.12.3 Bundle Attribute Functions

Short list of the functions:

- SetAttributes
- DeviceSetAttributes
- GetAttributes
- SetGlobalAttribute
- DeviceSetGlobalAttribute

16.1.12.4 Attribute and Bundle Definitions

The attributes for each attribute bundle type are defined below, following a general definition of colors, mixes and patterns.

Note: For area definitions, the area must be filled using the pattern that is current when the BEGIN AREA order is issued.

Colors

All colors will be passed as s32 values. They will either be indices into the logical color table, which will be stored by the engine, or 24 bit RGB values. All possible values will be passed to the engine.

Note that some special attribute values can be passed to the Engine:

-6 all color planes/bits '1'

-7 all color planes/bits '0'

-2 use the color given by index 7 (display) or index 0 (printer/plotter)

-1 use the color given by index 0 (display) or index 7 (printer/plotter)

-6 and -7 provide useful operands for BitBlt logical operations.

Attribute values of -1 and -2 will never loaded explicitly but will always produce the color value defined for index 0 or index 7 (given by the two rules above). If a color attribute of -1 and -2 has been set then -1 or -2 respectively will be returned in response to a query.

The range of color table indices is -2..MaxIndex.

For the default color table index -1 is black (same as 0 for a display, 7 for a printer) and index -2 is white (same as 7 for a display, 0 for a printer). *Note:* If the background/foreground mix is transparent, the background/foreground color will not be seen.

All values will be passed to the graphics engine, which will pass them unchanged to the device driver.

Foreground Mix Mode
Valid values are:

1 OR
2 Overpaint
3 Underpaint
4 Exclusive OR
5 Leave Alone (invisible)
6 AND
7 (inverse source) AND dest
8 source AND (inverse dest)
9 All zeros
10 Inverse (source OR dest)
11 Inverse (source XOR dest)
12 Inverse of dest
13 Source OR (inverse dest)
14 Inverse of source
15 (inverse source) OR dest
16 Inverse of (source AND dest)
17 All ones

Background Mix Mode
Valid values are:

1 OR
2 Overpaint
3 Underpaint
4 Exclusive OR
5 Leave Alone (invisible)

Support for the following foreground mixes is mandatory, other supported values may produce default mix (overpaint):

1 OR (for devices that are capable of supporting this)
2 Overpaint
5 Leave alone (invisible)

Support for the following background mixes is mandatory, other supported values may produce default mix (leave alone):

2 Overpaint
5 Leave alone (invisible)

The standard default background mix may be used in place of OR, Underpaint and Exclusive OR if these background mixes are not supported by the driver.

Patterns

pattern_symbol and line_pattern_symbol values in the range *1 thru 255* are valid.

Valid values of *line_pattern_symbol* and *pattern_symbol* in the base pattern set are:

1 thru 8 Solid shading with decreasing intensity.

9 Vertical lines
10 Horizontal lines
11 Diagonal lines 1, bottom left to top right
12 Diagonal lines 2, bottom left to top right
13 Diagonal lines 1, top left to bottom right
14 Diagonal lines 2, top left to bottom right
15 No shading
16 Solid shading
17 Every alternate pixel on

64 Blank

Note that solid shading *1* is less intense than solid shading *16*. Also, on many devices *17* may be distinct from the number *4* solid pattern and is useful for generating "Gray Text" among other things.

Pen (Line) Attributes

The line attribute bundle and 32 bit mask are defined as follows:

Bit Mask Number		
	RECORD LineAttributes,	
0	line_color	u32,
1	line_background_color	u32,
2	line_mix_mode	u16,
3	line_background_mix_mode	u16,
4	line_width	s32,
5	line_geometric_width	s32,
6	line_type	u16,
7	line_end	u16,
8	line_join	u16,

Note: Line_background_color and line_background_mix only

apply to line patterns. They are not used for the spaces in dot/dashed lines.

line_color
: Specifies the line foreground color.

line_background_color
: Specifies the line background color. This is only used for the line pattern, i.e., for thick lines where the pattern shading is non-solid, it is not used for the spaces in complex lines.

line_mix_mode
: Specifies the line foreground mix mode.

line_background_mix_mode
: Specifies the line background mix. This is only used for the line pattern, i.e., for thick lines where the pattern shading is non-solid, it is not used for the spaces in complex lines.

line_type
: Valid values for (cosmetic) *line_type* are:

```
1 Dotted
2 Short dashed
3 Dash, dot
4 Double dotted
5 Long dashed
6 Dash, double dot
7 Solid
8 Invisible
9 Every alternate pixel on

65 thru 254 User-defined line types
```

: Note that on many devices *9* may be distinct from the number *1* "Dotted" pattern.

line_width
: Specifies the cosmetic line thickness (used in all cases, except when the StrokePath option is specified on the ModifyPath function). This is treated as a 4-byte fixed point number with the high order word as the integer portion and the low order word as the fractional portion. Thus a value of 65536 specifies a width of 1.0.

: Valid values for *line_width* are:

```
10 Normal width
20 Thick (double width)
```

: Any other positive value is a multiplier on the 'normal' line width (values that are <= 1.0 may be treated as 'normal' and values > 1.0 as 'thick' by the

implementation). *Note:* Cosmetic line_ with attribute will be not be supported in the first release of Presentation Manager. If cosmetic line_ width is set to anything other than normal (default), a warning will be raised.

line_ geometric_ width
: Specifies the geometric line thickness in world coordinate space as an integer value. This is used only when the StrokePath option is specified on the ModifyPath function.

  Thick geometric lines will be treated as polygons and be transformed accordingly.

  Valid values for *line_ geometric_ width* are:

  >0 Thickness in world coordinates

line_ end
: Valid values are:

  1 Flat
  2 Square
  3 Round

line_ join
: Valid values are:

  1 Bevel
  2 Round
  3 Mitre

Pattern Attributes

The pattern attribute bundle and 32 bit mask are defined as follows:

Bit Mask Number	
	RECORD PatternAttributes,
0	pattern_color u32,
1	pattern_background_color u32,
2	pattern_mix_mode u16,
3	pattern_background_mix_mode u16,
4	pattern_set u16,
5	pattern_symbol u16;
6	pattern_origin(2) u32;

pattern_ color
: Specifies the pattern foreground color.

pattern_ background_ color
: Specifies the pattern background color.

pattern_mix_mode
: Specifies the pattern foreground mix mode.

pattern_background_mix_mode
: Specifies the pattern background mix mode.

pattern_set
: Specifies an lcid which identifies a symbol set or a bitmap. Valid values are:

'0001'X thru '00FE'X	`Loaded symbol set/bitmap identifier.`
'00F0'X	`Base pattern set identifier.`

pattern_symbol
: Specifies the identity of the required pattern in the current pattern symbol set (this attribute is ignored if the pattern set is a bitmap). If the value is outside the range of the symbol set, the standard default pattern is used.

  Values in the range *1 thru 255* are valid.

pattern_origin
: Specifies the pattern reference point for areas and thick lines.

  The pattern reference point is the point which the origin of the area filling pattern maps to. The pattern is mapped into the area to be filled by conceptually replicating the pattern definition in horizontal and vertical directions.

  Since the pattern reference point is subject to all of the transforms, if an area is moved by changing a transform and redrawing, the fill pattern will also appear 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. The pattern reference point is not subject to clipping, although of course the area to be filled will be.

Character Attributes

The character attribute bundle and 32 bit mask are defined as follows:

Bit Mask Number	
	RECORD CharacterAttributes,
0	char_color u32,
1	char_background_color u32,
2	char_mix_mode u16,
3	char_background_mix_mode u16,
4	char_set u16,
5	char_precision u16,
6	char_cell(2) s32,
7	char_angle_xy(2) s32,
8	char_shear_xy(2) s32,
9	char_text_align(2) s16,
10	char_spacing(2) s32,
11	char_break_extra(2) s32,
12	char_extra(2) s32,
13	char_direction u16;

char_color
: Specifies the foreground color.

char_background_color
: Specifies the background color.

char_mix_mode
: Specifies the foreground mix mode.

char_background_mix_mode
: Specifies the background mix mode.

char_set
: Specifies an lcid which identifies a symbol set or a font. If the lcid specifies an unloaded character set or a base symbol set, the engine uses the current code page and character precision to resolve which base symbol set to use. The code page (set by the function *SetCodePage*) identifies two base symbol sets, a vector and an image one. The value of *char_precision* determines which of these two sets is selected. Valid values for *char_set* are:

'0001'X thru '00FE'X	Loaded symbol set/font identifier.
'00F0'X	Base symbol set identifier.

char_precision
: Valid values for *precision* are:

1 Precision 1
2 Precision 2
3 Precision 3

char_cell

Specifies fixed point numbers for the width and height of a character cell in world coordinate space:

(s32_char_cell_width, s32_char_cell_height)

Each dimension is represented as a signed 4-byte integer, with a notional binary point between bits 16 and 15. Thus

+2.5 is represented by *'00028000'X* and
-2.5 is represented by *'FFFD8000'X*.

The width determines the spacing of consecutive characters along the baseline. Both width and height can be positive, negative or zero.

When either parameter is negative, the spacing occurs in the opposite direction to normal *and each character is drawn reflected* in character mode (precision 3). Thus, for example, a negative height in the standard direction in mode 3 means that the characters are drawn upside down, and the string drawn below the baseline (assuming no other transformations cause inversion).

A zero character width or height is also valid; here, the string of characters collapses into a line. If both are zero, the string is drawn as a single point.

char_angle_xy

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

In character mode (precision 1), the cell has no effect when characters are drawn.

In character mode (precision 2), the angle is used to determine the position of each character, but the orientation of characters within the character box is inherent in their definitions. The characters are positioned so that the lower left-hand corners of the character definitions are placed at the lower left-hand corners of the character boxes.

In character mode (precision 3), 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.

char_shear_xy

Specifies integer values that identify the end coordinates of a line originating at 0,0; the vertical strokes in subsequent character strings are drawn parallel to the defined line.

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 performed with the char_cell attribute).

It is an error to specify a zero value for *hy*, because this would imply an infinite shear.

char_text_align

Specifies the alignment, in horizontal and vertical directions, of subsequently output character strings:

(s16_horiz, s16_vert)

s16_horiz and s16_vert give the alignment of character strings horizontally and vertically. Together they define a reference point within the string that is positioned on the starting point specified for the string.

Possible values of *s16_horiz* are:

-1 Standard. The alignment depends on the current character direction:

Left to right (1)	`Left edge of first character`
Top to bottom (2)	`Left edge of first character`
Right to left (3)	`Right edge of first character`
Bottom to top (4)	`Left edge of first character`

1 Normal. The alignment depends on the current character direction:

Left to right (1)	`Left`
Top to bottom (2)	`Center`
Right to left (3)	`Right`
Bottom to top (4)	`Center`

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

3 Center alignment. The string is aligned on the arithmetic mean of left and right.

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

Possible values of *s16_vert* are:

-1 Standard. The alignment depends on the current character direction:

Left to right (1)	Bottom edge of first character
Top to bottom (2)	Top edge of first character
Right to left (3)	Bottom edge of first character
Bottom to top (4)	Bottom edge of first character

1 Normal. The alignment depends on the current character direction:

Left to right (1)	Base
Top to bottom (2)	Top
Right to left (3)	Base
Bottom to top (4)	Base

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

3 Cap alignment. The string is aligned on the cap of its topmost character. Where cap is not defined by the symbol set or font, this is the same as top.

4 Half alignment. The string is aligned on the arithmetic mean of base and cap.

5 Base alignment. The string is aligned on the base of its bottom character. Where base is not defined by the symbol set or font, this is the same as bottom.

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

The terms "top left", "bottom right", and so on, are well defined when the character angle and the direction of the coordinate system are such that the baseline is parallel to the x axis, running from left to right on the device, and there is no character shear.

If the character is rotated or sheared, the term "top left" applies to the corner of the character box that appears in the top left when no rotation or shear is applied. *Note:* The char_text_align attribute will be not be supported in the first release of Presentation Manager. If char_text_align is set to anything other than normal (default), a warning will be raised.

char_spacing

Specifies the amount of space or overlap to be provided between successive characters in a string, as two multipliers:

(s32_ char_ width_ mult, s32_ char_ height_ mult)

The multipliers apply to the *char_ cell* values. They give increments to the width and height which are applied equally to all characters in a string, irrespective of any proportional spacing or kerning which may take place.

Only one of the two multipliers is relevant for any particular character string primitive; this depends upon the value of *char_ direction.* The width increment is used for left-to-right and right-to-left character directions, and the height increment for top-to-bottom and bottom-to-top character directions.

char_ width_ mult and *char_ height_ mult* are 4_byte signed fixed point numbers with the high-order word as the integer portion and the low-order word as the fractional portion. Thus, a value of 65536 specifies a multiplier of 1.0.

The values may be negative, zero or positive:

- A negative value gives overlapping character cells.
- A value of zero results in standard spacing.
- A positive value allows extra space between character cells.

Note: The character spacing specified in the function *CharStringPos* is applied after the *char_ spacing* spacing as a final adjustment.

char_break_extra

Specifies the amount of additional space to be provided at the given break (normally space) character within a string:

(s32_ char_ width_ extra, s32_ char_ height_ extra)

This space is additional to any spacing produced by char_spacing and char_extra. The values char_width_extra and char_height_extra are additive deltas in world coordinates, and are irrespective of any proportional spacing or kerning which may take place.

Only one of the two values will be relevant for any particular character string primitive; this depends upon the character direction (char_direction). The width increment is used for left-to-right and right-to-

left character directions, and the height increment for bottom-to-top and top-to-bottom character directions.

char_width_extra, char_height_extra
: specify the width and height increments, respectively. These are each in the form of 4-byte signed fixed point numbers with the high-order word as the integer portion and the low-order word as the fractional portion. Thus, a value of 65536 specifies an increment of 1.0.

  The values may be negative, zero, or positive:

  - A negative value reduces the size of a break character
  - A value of zero (the standard default) gives the normal size (subject to any other spacing in force)
  - A positive value increases the size of a break character

char_extra
: Specifies the amount of space or overlap to be provided between successive characters in a string:

  (s32_char_width_extra, s32_char_height_extra)

  This gives the same spacing as char_spacing, except that the values are additive deltas rather than multipliers. If both character spacing and character extra is used, the effects will be cumulative.

  Only one of the two values will be relevant for any particular character string primitive; this depends upon the character direction (see char_direction). The width increment is used for left-to-right and right-to-left character directions, and the height increment for bottom-to-top and top-to-bottom character directions.

  char_width_extra, char_height_extra
  : specify the width and height increments, respectively. These are each in the form of 4-byte signed fixed point numbers with the high-order word as the integer portion and the low-order word as the fractional portion. Thus, a value of 65536 specifies an increment of 1.0.

    The values may be negative, zero, or positive:

- A negative value forces the characters closer together
- A value of zero (the standard default) results in standard spacing
- A positive value allows extra space between character boxes

char_direction
: Valid values for *direction* are:

1 Left to right
2 Top to bottom
3 Right to left
4 Bottom to top

If the specified direction is not valid, the default direction (Left to right) is used.

Image Attributes

The image attribute bundle and 32 bit mask are defined as follows:

Bit Mask Number	
	`RECORD ImageAttributes,`
0	`image_color u32,`
1	`image_background_color u32,`
2	`image_mix_mode u16,`
3	`image_background_mix_mode u16;`

image_color
: Specifies the image foreground color.

image_background_color
: Specifies the image background color.

image_mix_mode
: Specifies the image foreground mix mode.

image_background_mix_mode
: Specifies the image background mix mode.

Marker Attributes

The marker attribute bundle is defined as follows:

Bit Mask Number	
	`RECORD MarkerAttributes,`
0	`marker_color u32,`
1	`marker_background_color u32,`
2	`marker_mix_mode u16,`
3	`marker_background_mix_mode u16,`
4	`marker_set u16,`
5	`marker_symbol u16,`
6	`marker_cell(2) s32;`

marker_color
Specifies the marker foreground color.

marker_background_color
Specifies the marker background color.

marker_mix_mode
Specifies the marker foreground mix mode.

marker_background_mix_mode
Specifies the marker background mix mode.

marker_set
Specifies an lcid that identifies the required symbol set. Valid values are:

'0001'X thru '00FE'X Loaded symbol set identifier.

'00F0'X Base marker set identifier.

marker_symbol
Specifies the identity of the required marker symbol. The value identifies a symbol in the current marker set.

If the specified symbol is not valid, the standard default is used. Valid values in the base marker set are:

1 Cross
2 Plus
3 Diamond
4 Square
5 Six-point star
6 Eight-point star
7 Filled diamond
8 Filled square
9 Dot
10 Small circle

64 Blank

marker_cell
Specifies fixed point numbers for the width and height of a marker cell in world coordinate space:

(s32_marker_cell_width, s32_marker_cell_height)

Each dimension is represented as a signed 4-byte integer, with a notional binary point between bits 16 and 15. Thus

+2.5 is represented by '00028000'X and
-2.5 is represented by 'FFFD8000'X.

16.1.12.5 Function Definitions

SetAttributes (u32_BType, u32_DefsMask, u32_AttrsMask, p32_Attrs, u32_DcH, u32_FuncNo)

This sets attributes for the specified primitive type according to the defaults and attributes masks.

- Only attributes with their flag bit set in u32_AttrsMask are modified.
- Those attributes with their flag bit set in both u32_AttrsMask and u32_DefsMask are set to standard default value.
- Those attributes with their flag bit set in u32_AttrsMask only are set to the value specified by p32_Attrs.
- Those attributes with their flag bit set in u32_DefsMask only are unchanged.

Parameters:

u32_BType
: Specifies the bundle type as one of the following:

```
1   Pen (Line Attribute Bundle)
2   Character Attribute Bundle
3   Marker Attribute Bundle
4   Pattern Attribute Bundle
5   Image Attribute Bundle
```

u32_DefsMask
: Specifies the attributes to be set to their standard default values.

u32_AttrsMask
: Specifies the attributes to be modified.

p32_Attrs
: Points to the fixed format bundle record, specified above, containing the attribute values to be set, as specified by *u32_AttrsMask* In the record, only the attribute fields which correspond to the attribute flags set in u32_AttrsMask and not set in u32_DefsMask contain valid values.

Returns: *BOOL*

```
0 error
1 ok
```

DeviceSetAttributes (u32_BType, u32_DefsMask, u32_AttrsMask, p32_Attrs,u32_DcH,u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Attrs;
        ULONG   u32_AttrsMask;
        ULONG   u32_DefsMask;
        ULONG   u32_BType;
};
```

This sets attributes for the specified primitive type according to the defaults and attributes masks.

- Only attributes with their flag bit set in u32_AttrsMask are modified.
- Those attributes with their flag bit set in both u32_AttrsMask and u32_DefsMask are set to standard default value.
- Those attributes with their flag bit set in u32_AttrsMask only are set to the value specified by p32_Attrs.
- Those attributes with their flag bit set in u32_DefsMask only are unchanged.

Parameters:

u32_BType
: Specifies the bundle type as one of the following:

  1 Pen (Line Attribute Bundle)
  2 Character Attribute Bundle
  3 Marker Attribute Bundle
  4 Pattern Attribute Bundle
  5 Image Attribute Bundle

u32_DefsMask
: Specifies the attributes to be set to their standard default values.

u32_AttrsMask
: Specifies the attributes to be modified.

p32_Attrs
: Points to the fixed format bundle record, specified above, containing the attribute values to be set, as specified by *u32_AttrsMask* In the record, only the attribute fields which correspond to the attribute flags set in u32_AttrsMask and not set in u32_DefsMask contain valid values.

Returns: *BOOL*

0 error
1 ok

GetAttributes(u32_BType, u32_AttrsMask, p32_Attrs, u32_DcH, u32_FuncNo)

This returns the current value of the attributes specified in u32_AttrsMask. If the specified attribute is currently set to standard default value then the corresponding flag is set in the returned defaults mask value.

Parameters:

u32_BType
Specifies the bundle type as one of the following:

1 Pen (Line Attribute Bundle)
2 Character Attribute Bundle
3 Marker Attribute Bundle
4 Pattern Attribute Bundle
5 Image Attribute Bundle

u32_AttrsMask
Specifies the attributes to be returned. The mask contains a bit corresponding to each attribute in the bundle record, as defined above. For all the valid bits set to *1* in the mask, the corresponding attribute values and default mask bits are returned.

p32_Attrs
is a far pointer to a buffer in which the current attribute settings are returned. The only field that are updated are those corresponding to the attributes to be returned,as specified by the u32_AttrsMask.

Returns: *long int*

-1 error
>=0 defaults mask

Only those flags with the corresponding bit set in u32_AttrsMask will be updated, other flags are undefined.

SetGlobalAttribute(u32_AttributeType, u32_Attribute, u32_DcH, u32_FuncNo)

This sets the five individual primitive attributes to the specified value, in the pen, pattern, character, image and marker bundles.

Parameters:

u32_AttributeType
Specifies the attribute as one of the following:

1 Foreground Color
2 Background Color
3 Foreground Mix
4 Background Mix

u32_Attribute
Specifies the new value of the attribute.

Returns: *BOOL*

0 error
1 ok

DeviceSetGlobalAttributes
(u32_AttributeType,u32_Attribute,u32_DcH,u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Attribute;
        ULONG   u32_AttributeType;
};
```

This sets the five individual primitive attributes to the specified value, in the pen, pattern, character, image and marker bundles.

Parameters:

u32_AttributeType
Specifies the attribute as one of the following:

1 Foreground Color
2 Background Color
3 Foreground Mix
4 Background Mix

u32_Attribute
Specifies the new value of the attribute.

Returns: *BOOL*

0 error
1 ok

16.1.12.6 Drawing Functions

These functions pass individual drawing orders to the Graphics Engine. The engine will then draw, correlate and/or take bounds on the drawing primitives, as determined by the current values of the draw, correlate and bounds flags.

The engine is assumed to clip to the appropriate part of the window; that is the region excluding any window border or frills.

Coordinates are passed as signed 32 bit numbers in a logical space called WCS (world coordinate space).

Angles are passed as signed 32 bit numbers. A value of 2**31 means a full circle. A value of zero refers to a direction along the positive x axis. Positive values are taken as being anticlockwise from the positive x axis, so that, for example, 2**29 refers to a direction along the positive y axis.

16.1.12.7 'Move' Type Orders

It is intended that series of line, arc and fillet orders should all join up correctly, including the on/off counts according to the current line style/type. The rules given below allow the smooth joining up of a series of lines, arcs, and fillets. Such a series will start and end at the expected positions. The only case they fail on is that of a closed figure drawn in XOR mode, when the first (and last) pel will be drawn twice, and therefore lost.

It is also important that it is clearly specified which orders cause an implicit figure closure during an area.

Certain orders are defined to be 'move' type operations. A move causes three things to happen:

- The 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 (the current device coordinate position is not the same as the device coordinate position the figure was started at), then an implicit closure line is drawn to close it.

Subsequent 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 full arc, box or pie slice drawn with boundary, (that is any closed figure) will be drawn with its boundary complete (no missing pels) and with the line pattern sequence honored around all the parts of its boundary. To allow construction of complex area boundaries, such closed figures are not considered to be 'move' type operations.

The following engine functions are defined to be 'move' type operations.

- 'Set Current Position'.
- Any 'Set' that changes (or might change) the transform from WCS to device coordinates, for instance 'Set Model Transform', or 'Set Window/Viewport Transform'.

- Any 'Set' that changes (or might change) the current clipping, for instance 'Set Viewing Limits'.

Refer to u32_FuncNo page 5-1 and 6-7 for further details. *Note:* a different set of rules is needed if constructing a boundary for scan line area filling, for example, you may choose to ignore line style, and draw all lines solid, with first pel off, last pel on. This boundary is of course different from the boundary that is drawn on the screen after the interior is filled.

16.1.12.8 Filled Closed Figures

Certain closed figures can be specified as being filled, outline, or both, or neither. The filling of these figures (box, full arc, and pie slice) will use the current pattern (area) attributes. The filled version of these figures can be treated as an error if already in an area definition.

16.1.12.9 Correlation on Areas

Correlation obeys the u32_FuncNo rules. In the case of correlating on areas this is particularly complex.

- SET CURRENT POSITION and END AREA will 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.
- The area interior itself can cause a correlation hit, which must be reported on the END AREA order. In addition, a double hit can occur on END AREA. This occurs when
  - the end area causes an area closure line to be drawn,
  - the boundary is being drawn

    and
  - the closure line and the area interior both intersect the pick window.

  In this case a special double hit return code is passed back.
- The lines (and 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 implies that some work must be done, as well as just journalling the calls that define the area boundary.

16.1.12.10 Correlation on Strokes

A hit is returned on the the nominal width strokes as they come in, and then correlation is performed on the whole figure on EndStrokes, much like areas.

16.1.12.11 Transform Matrix Precision

Matrix elements are represented as a 16 bit signed integer and a 16 bit fractional part. These precision limits apply during engine matrix multiplication for all initial, intermediate and final matrix element values.

16.1.12.12 Code Page

It is assumed in this section that the engine (or device driver) will be able to tell from the selected font or symbol set whether it is a single or double byte character set, and draw the characters appropriately.

16.1.12.13 List of Functions

Short List of Function Calls:

- Arc
- BeginArea
- BeginPath
- Box (BoxInterior, BoxBoundary, BoxBoth)
- CharString
- CharStringPos
- CloseFigure
- CombinePaths
- DeletePath
- DrawFrame
- EndArea
- EndPath
- FillPath
- FullArc (FullArcInterior, FullArcBoundary, FullArcBoth)
- ImageData

- ModifyPath
- OutlinePath
- QueryCharPositions
- PartialArc
- Poly (PolyFillet, PolyLine, PolyMarker, PolySpline)
- PolyFilletSharp
- PolyShortLine
- QueryClipPath
- QueryTextBox
- QueryTextBreak
- SelectClipPath

Arc(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

Creates an arc, using the current arc parameters, through three x,y positions starting at the current x,y position.

Parameters:

p32_xy Points to

```
        s32_spare    (may be used as work area)
        s32_spare    (may be used as work area)
        s32_x1       (coordinates of second point)
        s32_y1
        s32_x2       (coordinates of third, and final, point)
        s32_y2
```

Returns: *short int*

```
0 error
1 ok
2 CorrelateHit
```

Upon completion, the current x,y position is the third position of the arc.

BeginArea(u32_flags, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Flags;
};
```

Indicates the beginning of a set of primitives that define the boundary of an area.

Only certain drawing functions may be used to build the boundary of an area, specifically those that draw lines or arcs. Functions that draw character strings, markers, images, or BitBlts are not allowed in an area definition.

Parameters:

u32_flags

Specifies whether boundary lines are to be drawn, and what algorithm is to be used to determine the area interior.

Bit 0	Set to 1 to draw boundary lines
Bit 1	Set to 1 for winding fill mode, set to 0 for alternate fill mode

Although the current x,y position is not changed by BeginArea, it will be affected by the drawing orders in the boundary definition.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

BeginPath(s32_PathId, u32_Options, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
   ULONG  u32_Options;
   ULONG  s32_PathId;
};
```

Starts a path definition and gives it an identifier. The path may be either a one-dimensional (PATH_1D) or a two-dimensional (PATH_2D) path.

A path, whether it is a PATH_1D or a PATH_2D type path, is defined in terms of a number of figures. A figure is defined by

line and/or curve functions, and is separated from other figures by a CloseFigure function, or (for a PATH_1D type path only) a "move" function.

If a figure in a PATH_1D path definition is terminated by a "move" type function, or by EndPath, then that figure is said to be *open.* Otherwise (i.e., it is terminated by CloseFigure), it is *closed.* All figures contained within a PATH_2D path definition must be *closed.*

A PATH_2D type path may be defined using either ALTERNATE or WINDING mode (see BeginArea).

Path definition functions are terminated by EndPath. If there are no primitives between BeginPath and EndPath, a null path will be defined.

Once a path has been defined, the definition cannot be reopened. An attempt to define a path with an identifier for which a path definition already exists, results in that definition being replaced.

A path definition is bound in device coordinates at the time the path is defined. Any subsequent change to any transforms (other than to the final translation if the window is moved) has no effect on the path.

The path descriptions which are built as a result of the path definitions contain, for both types of path (PATH_1D and PATH_2D), information about the outline of the path, which was defined by the line and curve functions in the definition. Additionally, a PATH_2D path description will contain information describing the interior "area" defined by the path.

The following are the only primitive/attribute functions allowed within a path definition:

```
SetModelXform
SetCurrentPosition
GetCurrentPosition
PolyLine
Box
SetArcParams
GetArcParams
Arc
FullArc
PartialArc
PolySpline
PolyFillet
PolyFilletSharp
CharString
CharStringPos
```

The CharStringxxx functions are allowed only if the current font is an outline font. Note that character attribute setting functions are not allowed.

It is invalid for this function to occur within an area definition.

A single path can not contain more than 64K of data.

Parameters:

s32_PathId
: Path identifier. Valid values are in the range 1..64.

u32_Options
: Path generation options:

  1 PATH_1D
  : A one-dimensional path is to be defined.

  2 PATH_2DALTERNATE
  : A two-dimensional path is to be defined, in alternate mode.

  3 PATH_2DWINDING
  : A two-dimensional path is to be defined, in winding mode.

Returns: *BOOL*

0 Error
1 OK

Box(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

Draws a rectangular box with one corner at the current x,y position and the other at the point specified. The sides of the box (before transformation) are parallel to the x and y axes.

The corners of the box may be rounded by means of quarter ellipses of the specified diameters. If the value of either diameter is zero, then no rounding occurs. If the value of either diameter exceeds the length of the corresponding side, then that length is used as the diameter instead.

The box may be filled, or just an outline, or both. This is achieved by using different function numbers.

Parameters:

p32_xy Long pointer to parameters defining the box as follows:

```
s32_spare     May be used as work area
s32_spare     May be used as work area
s32_cornerX   X coordinate of second corner of box
s32_cornerY   Y coordinate of second corner of box
s32_Xdiam     X diameter of ellipse used to round
              corners
s32_Ydiam     Y diameter of ellipse used to round
              corners
```

The current x,y position is not altered by Box.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

CharString(s32_n, p32_ch, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_ch;
        ULONG   s32_n;
};
```

Draws a character string starting at the current x,y position.

Parameters:

s32_n Specifies the number of bytes in the character string.

p32_ch Long pointer to the string of character codepoints.

The current x,y position is moved to the point at which the next character string would have been drawn, had there been one.

The Gpi function GpiVectorSymbol is provided to interpret a vector symbol.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

CharStringPos(p32_xy, u32_options, s32_n, p32_ch, p32_dx, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Dx;
        ULONG*  p32_Ch;
        ULONG   s32_N;
        ULONG   u32_options;
        ULONG*  p32_xy;
};
```

Draws a character string starting at the current x,y position.

A vector of increments may optionally be specified. which allows control over the positioning of each character after the first. These are distances measured in world coordinates (along the baseline for left-to-right and right-to-left character directions, and along the shearline for top-to-bottom and bottom-to-top). The i'th increment is the distance of the reference point (e.g., bottom left corner) of the (i+1)'th character from the reference point of the i'th. The last increment may be needed to update current position.

These increments, if specified, set the widths of each character. Any spacing called for by

- char_spacing
- char_extra
- char_break_extra

is applied in addition to the widths defined by the vector.

A further option allows a rectangle to be specified, which is to be used as the background of the string, rather than using the normal method of defining the background. This rectangle will be painted using the current character background color and an overpaint mix. Both corners of the rectangle are specified, so that the rectangle is positioned independently of current position.

A further option allows clipping of the string to the rectangle. This is independent of whether the rectangle is actually drawn.

The string may optionally be drawn de-emphasized. This is used in menus to denote options which are not currently selectable. The implementation may choose whichever method of de-emphasis is most appropriate on the particular device.

Current position may optionally be updated to the point at which the next character would have been drawn, had there been one, or it can be left unchanged by this function.

Parameters:

p32_xy Long pointer to the an array of six coordinates, defining the opaque rectangle. The first two coordinates of this array will be spare and the remainder will specify two corners of the rectangle that defines the background of the characters (ignored if both the opaque and clip option flags are zero). The opaque rectangle (drawn if bit 0 below = 1) will ignore any background mix attributes, and be drawn using overpaint and the character background color attribute.

u32_options
Flags controlling the function as follows:

```
Bit 0   = 0 Do not opaque
        = 1 Opaque rectangle

Bit 1   = 0 Increment vector not present
        = 1 Increment vector present

Bit 2   = 0 Normal text
        = 1 Grayed (de-emphasized) text

Bit 3   = 0 Move current position to end of
            character string
        = 1 Leave current position unchanged

Bit 4   = 0 Do not clip
        = 1 Clip string to rectangle

Note - if rectangle not present, then bits 0 & 4
       must be zero
```

s32_n Specifies the number of characters (bytes) in the character string.

p32_ch Long pointer to the string of character codepoints.

p32_dx
Long pointer to an array of s32 numbers, the character increments. There will be s32_n numbers in this array. They are given in world coordinates.

The current x,y position may optionally be moved to the end of the character string, according to the u32_options.

The Gpi function GpiVectorSymbol is provided to interpret a vector symbol.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

CloseFigure(u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

Closes a figure within a path definition. The current figure is closed by means of a line drawn to the start point of the figure.

This function must be used to close each figure within a PATH_2D path. Within a PATH_1D path it is optional between figures; an "open" figure may be generated by starting a new figure (with a "move" type function), or by ending the path, without first issuing CloseFigure.

If this function occurs outside a path definition, it has no effect.

Returns: *BOOL*

0 Error
1 OK

CombinePaths(s32_DestPathId, s32_Src1PathId, s32_Src2PathId, u32_Mode, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
   ULONG  u32_Mode;
   ULONG  s32_Src2PathId;
   ULONG  s32_Src1PathId;
   ULONG  s32_DestPathId;
};
```

Combines two paths. The result becomes the definition of the destination path, which may or may not already exist. It may also be the same as one of the source paths.

Various combination modes are allowed, depending upon whether the paths to be combined are PATH_1D or PATH_2D types. Disallowed modes give an error.

A single path can not contain more than 64K of data.

Parameters:

s32_DestPathId
: Identifier of the destination path. It must be > 0.

s32_Src1PathId, s32_Src2PathId
: Identifiers of the source paths. They must be > 0 (except where s32_Src2PathId is ignored, see below).

u32_Mode

Method of combination of paths:

Src1 is PATH_2D, Src2 is PATH_2D (or ignored)
Dest will become a PATH_2D path:

```
CPATH_OR     (1) - Union of Src1 and Src2
CPATH_COPY   (2) - Src1 only copied to Dest (Src2 ignored)
CPATH_XOR    (4) - Symmetric difference of Src1 and Src2
CPATH_AND    (6) - Intersection of Src1 and Src2
CPATH_DIFF   (7) - Src1 AND NOT(Src2)
```

Src1 is PATH_1D, Src2 is PATH_1D (or ignored)
Dest will become a PATH_1D path:

```
CPATH_OR     (1) - Append Src2 to Src1
CPATH_COPY   (2) - Src1 only copied to Dest (Src2 ignored)
```

Src1 is PATH_1D, Src2 is PATH_2D
dest will become a PATH_1D path:

```
CPATH_AND    (6) - Src1 clipped to Src2
CPATH_DIFF   (7) - Src1 clipped to the inverse of Src2
```

Returns: *BOOL*

0 Error
1 OK

DeletePath(s32_PathId, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
   ULONG  s32_PathId;
};
```

Deletes a path definition, and all of its storage, from the current level of the Device Context (if the path is saved below it remains in existence below). It is an error if the path is currently selected as a clip path.

Parameters:

s32_PathId

Path identifier.

A value of -1 indicates that all path definitions in the specified Device Context are to be deleted (this is an error if 'any' clip path is currently selected.

Returns: *BOOL*

0 Error
1 OK

DrawFrame(p32_Rect, p32_xy, u32_Options, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Options;
        ULONG   p32_xy;
        ULONG   p32_Rect;
};
```

This draws a rectangle surrounded by a frame. The interior is drawn with the pattern attributes, and the border is drawn with the line attributes.

The coordinates passed are in DC origin device coordinates.

This is a required function for display device drivers. It is used by the user interface to improve the performance of wide border dragging and dialog box posting.

Parameters:

p32_Rect
: A long pointer to a rectangle. This rectangle completely surrounds the drawing, i.e., the border is inside the rectangle.

p32_XY
: A long pointer to the width of the sides, and the height of the top/bottom, of the border.

u32_options

```
Bit 0 - 0 draw the interior
      - 1 do not draw the interior
```

EndArea(u32_cancel, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Cancel;
};
```

Indicates the end of a set of primitives that define the boundary of an area.

If there is a correlation hit on (any part of) the area interior it is returned on this function. (Correlation hits on the boundary are returned on the primitive causing the hit.)

If the current figure is not closed, this function will generate a closure line from the current position to the start of the current figure. If a correlation hit is diagnosed on this line as well as on the area interior, a special return code indicates this double hit.

On devices with hardware assist for area fill (such as an area fill plane), this facility may be used, or the area definition may be built up in an area fill plane in ordinary PC storage. In the case of convex figures, there may be a performance gain in just recording the start and end pel position across each scan line. Whatever algorithms are used, it is crucial that the area interior should be filled identically in each case, otherwise bit map operations may fail to join correctly when copied to the screen, etc. This is obviously particularly crucial when the area is being dragged around the screen using a mix mode of XOR to be able to remove it.

Upon completion, the current x,y position is the last x,y position specified in the boundary definition, unless figure closure occurred, in which case it is the start of the last figure in the area definition.

Parameters:

u32_cancel
: If this is 0, the area is to be drawn. If it is 1, the area is to be cancelled (terminated without being drawn). An EndArea(Cancel) without a previous BeginArea is valid but has no effect. Thus it is possible to reset the area bracket to a known state without any knowledge of its current state.

Returns: *short int*

0 error
1 ok
2 CorrelateHit
3 MultipleCorrelateHit

EndPath(u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

Ends the definition of the current path.

Returns: *BOOL*

0 error
1 ok

FillPath(s32_PathId, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
   ULONG  s32_PathId;
};
```

This takes a PATH_2D path, and draws the interior of the path using the current pattern attributes.

Parameters:

s32_PathId
: The identifier of the path whose interior is to be drawn. It must be > 0 and specify a PATH_2D.

Returns: *BOOL*

```
0 Error
1 OK
2 Correlate hit
```

FullArc(u32_m, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_M;
};
```

Creates a full arc with its center at the current x,y position.

The full arc may be filled, or just an outline, or both, or neither. This is achieved by using different function numbers.

Whether the full arc is drawn clockwise or counterclockwise is determined by the arc parameters.

Parameters:

u32_m
: Specifies the multiplier that determines the size of the arc in relation to an arc with the current arc parameters. The value passed is treated as a 4-byte fixed-point number with the high-order word as the integer portion, and the low-order word as the fractional portion. Thus, a value of 65536 specifies a multiplier of 1.

Returns: *short int*

```
0 error
1 ok
2 CorrelateHit
```

The current x,y position is not changed by FullArc.

ImageData(p32_data, s32_n, s32_row, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   s32_Row;
        ULONG   s32_N;
        ULONG*  p32_Data;
};
```

Draws a row of image data.

Separate calls are required for each row of image data. The data is written on adjacent rows starting at the top row in the image area.

The implementation should not assume that unused bits in the last byte of data for a row are zero.

Parameters:

p32_data
: Points to a string of image data with one bit per pel.

s32_n
: Specifies the number of data bits that are required in the drawing order.

s32_row
: Specifies the row number of the image data. Row 0 is the same row as the current position, row 1 is the next one down the screen, and so on.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

The current position is not affected by this order.

ModifyPath(s32_PathId, u32_Mode, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
   ULONG  u32_Mode;
   ULONG  s32_PathId;
};
```

Modifies the specified path in one of the ways described below.

A single path can not contain more than 64K of data.

Parameters:

s32_PathId
: Identifier of the path to be modified. It must be > 0.

u32_Mode
: Modification required, as follows:

1 BOUNDRY
: PATH_2D to PATH_2D

Converts the path to one describing the same interior area, but which no longer contains any of the lines or curves which lie wholly within the area. This modification will normally be performed on a PATH_2D path which was initially defined using WINDING mode, so that if a OutlinePath operation is performed on it, these interior lines will not be drawn.

2 INVERT
: PATH_2D to PATH_2D

Inverts the path, so that what was outside the path is now inside it, and vice versa.

3 CLIP_TO_CURRENT
: PATH_1D to PATH_1D or PATH_2D to PATH_2D

Clips the path to the current Clip Path.

The path is flagged as having been clipped. If a path is drawn, it will be clipped to the current clip path, unless this flag has been set for the path.

In addition to clipping, this function is a hint to the system to cache other data (such as the PolyScanLines data created in filling) which may be used repeatedly.

If no clip path has been selected, this function has no effect, other than the performance hint mentioned above.

4 2DWINDING
: PATH_1D to PATH_2D

Converts a PATH_1D path to a PATH_2D path, using the WINDING method (see BeginArea). Any unclosed figures in the original path are discarded without warning.

5 2DALTERNATE
PATH_1D to PATH_2D

Converts a PATH_1D path to a PATH_2D path, using the ALTERNATE method (see BeginArea). Any unclosed figures in the original path are discarded without warning.

6 STROKEPATH
PATH_1D or PATH_2D to PATH_2D

Converts the path to a PATH_2D one which describes the envelope of a wide line, stroked using the current geometric wide line attribute. *Note:* that this is the only function which can cause geometric wide lines to be constructed.

The envelope will include the effects of line joins, and line ends, according to the current values of these attributes. The following should be noted:

- A line may be joined to, for example, an arc. The common point will be handled according to the Line Join parameter, rather than applying Line Ends at each end.
- If a figure has been closed using GpiCloseFigure, then the joining rules will be followed rather than the ending rules, at the start/end point.
- The envelope will take account of any crossings, so that, for example, a stroked "X" will not, if subsequently drawn in exclusive_OR mode, have a hole in the middle.

Returns: *BOOL*

0 Error
1 OK

OutlinePath(s32_PathId, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
   ULONG  s32_PathId;
};
```

This takes either a PATH_1D or a PATH_2D path, and draws the outline using the current line attributes, including the cosmetic line width, but not the geometric line width.

Parameters:

s32_PathId
: The identifier of the path to be stroked. It must be > 0.

Returns: *BOOL*

0 Error
1 OK
2 Correlate hit

QueryCharPositions (u32_options, s32_n, p32_ch, p32_vector, p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
   ULONG      s32_vector;
   ULONG*     p32_ch;
   ULONG      s32_n;
   ULONG      u32_options;
};
```

Returns the positions in world coordinates of where the currently associated device will place each given character, taking into account kerning, extra space, etc.

A vector of increments may optionally be specified, which allows control over the positioning od each character after the first. These distances are measured in world coordinates(along the baseline for left-to-right and right-to-left character directions, and along the shearline for top-to-bottom and bottom-to-top).

These increments, if specified, set the widths of each character. Any spacing called for by

char_spacing

char_extra

char_break_extra

is applied in addition to the widths specified by the vector.

Parameters:

u32_options
: Flags controlling the function as follows:

```
                Bit 0       = 0 Reserved

CHS_VECTOR (Bit 1)          = 0 Increment vector not present
                            = 1 Increment vector present

                Bits 2-31   = 0 Reserved
```

s32_n Specifies the number of bytes in the character string.

p32_ch Long pointer to the string of character codepoints.

p32_vector
A vector of (n values of) increment values. These are 4-byte signed integers in world coordinates.

p32_xy Long pointer to an array of n + 1 positions. The first element of the array contains the current position and the last contains the new current position.

Returns: *BOOL*

0 error
1 ok

PartialArc(p32_xy, u32_m, s32_ts, s32_te, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   s32_Te;
        ULONG   s32_Ts;
        ULONG   u32_M;
        ULONG*  p32_xy
};
```

Draws two figures:

1. A straight line, from current position to the starting point of a partial arc, and
2. The arc itself, with its center at the specified point.

The full arc, of which the arc is a part, is identical to that defined by FullArc. The part of the arc drawn by this primitive is defined by the parameters s32_ts and s32_te, which represent the angles subtended from the center, if the current arc parameters specify a circular form. If they do not, these angles are skewed to the same degree that the ellipse is a skewed circle. s32_ts and s32_te are measured anticlockwise from the x axis of the circle prior to the application of the arc parameters.

Whether the arc is drawn clockwise or anticlockwise is determined by the arc parameters, s32_ts and s32_te.

Parameters:

p32_xy A pointer to an x,y coordinate pair which are the coordinates of the center of the arc.

u32_m Specifies the multiplier that determines the size of the arc in relation to an arc with the current arc parameters. The value passed is treated as a 4-byte fixed-point number with the high-order word as the integer

portion, and the low-order word as the fractional portion. Thus, a value of 65536 specifies a multiplier of 1.

s32_ts, s32_te
: Specify the start and ending angles.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

Upon completion, the current x,y position is set to the final point of the arc.

Poly(p32_xy, s32_n, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   s32_N;
        ULONG*  p32_xy;
};
```

Creates a poly primitive, which can be a fillet, spline, line or marker, starting at the current position, using the array of x,y coordinate pairs passed. Different values of u32_FuncNo will be used for the different kinds of poly primitive.

The definition of a PolyFillet is as follows:

Creates a fillet on a series of connected lines, with the first line starting at current position. If only two points are supplied, an imaginary line is drawn from current position to the first point, and a second line from the first point to the second. A curve is then constructed, starting at current position and tangential to the first line at that point. The curve is drawn such that it reaches the last point at a tangent to the second line. The curve has the appearance of a fillet. The lines are imaginary, and are not drawn. If more than two points are supplied, an imaginary series of lines is constructed through them (as in PolyLine). All the lines except the first and last are then divided in two at their mid-points. A series of curved fillets are then drawn, each starting at the end point of the last, at one of the mid-points.

The definition of a PolySpline is as follows:

Creates a succession of Bezier splines. The first one starts from current position and goes to the third specified point, with the first and second points used as control points. Subsequent splines start from the ending point of the previous spline, and end at the next specified point but two, with the intervening points their first and second control points. The number of x,y pairs in the array must be 3*s where s is the number of splines

The x,y pairs are given in the following order:

```
     c11, c12, e1, c21, c22, e2, c31, c32, e3, ...
where
     csi       is the i'th control point of the s'th spline
     es        is the endpoint of the s'th spline
```

Parameters:

p32_xy Points to an array of x,y coordinates.

An extra x,y pair will be passed at the start of the xy array (and not included in the count), as work space. The whole array may, if need be, be overwritten by the transformed coordinates.

s32_n Specifies the number of x,y pairs. *Note:* For PolyLine, s32_n = 0 is valid and the function is then ignored.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

Upon completion, the current x,y position is the last line in the series.

PolyFilletSharp(p32_xy, s32_n, p32_s, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_S;
        ULONG   s32_N;
        ULONG*  p32_xy;
};
```

Creates a fillet on a series of connected lines, with the first line starting at current position. Subsequent points identify the end points of the lines.

The first fillet is drawn using the two lines, one from current position to the first (x,y) point specified (its control point), and one from there to the second point specified. The fillet starts from current position, and ends at the second point specified. It is tangential to the first line at current position, and to the second line at the second specified point. The sharpness of this fillet is given by the first sharpness value.

Each subsequent fillet is drawn starting from the end point of the previous fillet, and uses the next two lines in the sequence, in a similar way. Thus two points and one sharpness value are specified for each fillet.

Note that this function differs from PolyFillet in the following ways:

- The sharpness of each fillet is explicitly specified.
- Both the control and the end point of each fillet are explicitly specified.
- Adjacent fillets will in general have a discontinuity in gradient, unless the points are chosen so that this is not the case.

The sharpness of each fillet is defined as follows. Let A and C be the start and end points, respectively, of the fillet, and let B be the control point. Let W be the mid-point of AC. Let D be the point where the fillet intersects WB. Then the sharpness is given by

```
sharpness = WD/DB
```

If

```
S > 1.0, a hyperbola is drawn
S = 1.0, a parabola is drawn
S > 1.0, an ellipse is drawn
```

Parameters:

p32_xy Points to an array of x,y coordinates.

An extra x,y pair will be passed at the start of the xy array (and not included in the count), as work space. The whole array may, if need be, be overwritten by the transformed coordinates.

s32_n Specifies the number of x,y pairs.

p32_s Specifies the sharpness of the fillets. It is a far pointer to an array of n/2 elements, each array element being an s32_sharpness parameter. Each value, when divided by 65536, gives the sharpness of successive fillets.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

Upon completion, the current x,y position is the last point in the series.

PolyShortLine(p32_data, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Data;
};
```

Draws a set of lines encoded as a series of steps. This call is not revealed in the API.

Parameters:

p32_Data
: Is a pointer to a polyshort structure, of the following format:

```
struct POLYSHORT {
    ULONG  s16_StartX
    ULONG  s16_StartY
    ULONG  s16_EndX
    ULONG  s16_EndY
    ULONG  s16_Count
    ULONG* p32_Steps
};
```

where:

s16_StartX, s16_StartY
: The start point of the polyshortline

s16_EndX, s16_EndY
: The end point of the polyshortline

s16_Count
: The number of bytes of data pointed to by p32_Steps

p32_Steps
: Points to a byte array of encoded lines. Each line is encoded as follows:

```
Bits 2:0   Direction to draw:

                0 1 2
                 \|/
                7-x-3
                 /|\
                6 5 4

Bit 3      Draw/Skip
             0 => Draw these pels
             1 => Skip these pels

Bits 7:4   Number of pels to draw (1 to 16)
```

The current position is not affected by this call. The lines are assumed already clipped.

QueryClipPath(u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This returns the identifier of the currently selected clip path, if any.

Returns: *LONG*

```
-1 Error
 0 Null (no path was previously selected)
>0 Identifier of previously selected path
```

QueryTextBox (s32_n, p32_ch, s32_count, p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
        ULONG   s32_Count;
        ULONG*  p32_Ch;
        ULONG   s32_N;
};
```

This processes the specified string as if it were to be drawn, using the current character attributes, and returns an array of up to 5 x,y coordinate pairs. The first four of these are the coordinates of the top-left, bottom-left, top-right and bottom-right corners of the parallelogram which encompasses the string when drawn on the associated device. The fifth point is the concatenation point, that is the position at which a subsequent string would have to be drawn if it were to follow on smoothly.

All coordinates are relative to the start point of the string, as defined by the character direction.

Parameters:

s32_n Specifies the number of bytes in the character string.

p32_ch Long pointer to the string of character codepoints.

s32_count
Specifies the number of x,y pairs in the p32_xy array.

p32_xy Long pointer to the return array of 5 x,y pairs.

Returns: *BOOL*

```
0 error
1 ok
```

QueryTextBreak (s32_n, p32_ch, s32_len, p32_n, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_N;
        ULONG   s32_Len;
        ULONG*  p32_Ch;
        ULONG   s32_N;
};
```

This processes the specified string as if it were to be drawn, using the current character attributes, and finds where the string must be split if it is not to exceed the specified extent.

Parameters:

s32_n Specifies the number of bytes in the character string.

p32_ch Long pointer to the string of character codepoints.

s32_len Specifies the maximum extent of the string, measured along the baseline for left to right or right to left character directions, and along the shear line for top to bottom or bottom to top character directions.

p32_n Long pointer to a s32 variable to return the number of characters which fit into the extent. If no characters fit, zero is returned.

Returns: *BOOL*

```
-1 error
>=0 extent unused by p32_n characters in world coordinates
```

SelectClipPath(s32_PathId, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
   ULONG  s32_PathId;
};
```

This takes a PATH_2D path, and selects it to be the current clip path. The identifier of the path which was previously selected as the clip path (if any) is returned.

The clip path (bound in device coordinates when the path was defined, as usual) is used for all subsequent drawing, except for OutlinePath or FillPath, where the path is already flagged as having been clipped (see ModifyPath).

While a path is selected as the clip path, it may not be modified (though it may be used for a read-only operation, such as FillPath, or as the source of a CombinePath operation).

Parameters:

s32_PathId
: If this is > 0, it is the identifier of the path to be selected (which must be a PATH_2D path). If 0 is specified, any path clipping is removed.

Returns: *LONG*

```
-1 Error
 0 Null (no path was previously selected)
>0 Identifier of previously selected path
```

16.1.12.14 AVIO Functions

Short List of Function Calls:

- CharRect
- CharStr
- ScrollRect
- UpdatePointer

Note: It will be an applications responsibility to ensure that windows containing alphanumeric data are device cell aligned if appropriate. It will be a property of a device driver whether non-cell aligned characters will be visible.

All column, row, length, width and height values correspond to cells within a presentation space buffer.

The origin of the presentation is assumed to be at the bottom left, i.e., row column = (0,0) corresponds to the the character cell in the bottom left hand corner.

This interface is written on the assumption that the graphics engine performs any necessary clipping of the alphanumeric data into a window.

All alphanumeric engine calls are passed a pointer to a Vio presentation space. From this the engine extracts current state data as needed to allow it to update an output device.

A Vio presentation space contains:

- the device context handle for the presentation space.
- a RAM semaphore to control access to the presentation space from the Shield Layer and the Vio subsystem.
- a near pointer to a private area for use by the Shield Layer and the Graphics Engine.
- a near pointer to the buffer containing the character, attribute cells to be output.
- the size of the character cell buffer (rows,columns).
- the number of bytes in a buffer cell, currently either 2 (CGA format buffer) or 4 (extended CGA format).
- a structure containing two variables for the number of pels in a device cell (height,width). This structure is set by the Vio subsystem when the presentation space is created to the first cell size returned by QueryDeviceCaps. The Graphics Engine will ensure that it uses a base font to match this cell size. Unpredictable results will occur if a loaded font doesn't match this cell size.
- structures for the cursor position and state (zero visibility means the cursor is invisible).
- the origin is a structure which identifies the cell (row,column) in the character cell buffer to be drawn in the top left hand corner of the window.
  - For the Engine, the Window origin coordinates in the PS:
    - are relative to the bottom left corner of the LVB.
    - define the cell which is displayed at the bottom left corner of the client rectangle.
  - The transformation for converting from the top-left origin used for VioSetOrigin to the bottom-left origin used in the PS is:

    ```
    BL_Origin_Row=lpVioPS->BufferRowCount - lpVioPS->WindowHeight -
    TL_Origin_Row;
    ```

  - The internal interfaces to GreCharRect, GreCharStr, and GreScrollRect the co-ordinates passed are changed such that:
    - They are relative to the bottom left corner of the LVB.
    - For rectangle descriptors they define the bottom left corner of the rectangle being described.
  - For a rectangle descriptor the coordinate transformation is:

    ```
    BL_Row_Coordinate=lpVioPS->BufferRowCount - RectangleHeight -
    TL_Row_Coordinate;
    ```

- For the string descriptor passed to GreCharStr the transform is:

```
BL_Row_Coordinate=lpVioPS->BufferRowCount - 1 - TL_Row_Coordinate;
```

- The string case is like a rectangle with one row.

- further variables as required by the Vio subsystem.

Each time the presentation space character cell size changes, a null CharRect call will be issued to the engine (i.e., with u32_StartRow, u32_StartCol, u32_RectWidth and u32_RectHeight all set to zero).

Buffer cell byte contents:

- Code point
- CGA Attribute byte

```
bit 7-4 background color
bit 3-0 foreground color
each 4 bit color value will correspond to some explicit 24 bit
RGB value. The RGB values are assumed to be predefined and
fixed within the graphics engine. They should match the colors
available on a CGA.
```

- Extended Attribute byte (4 byte cells only)

```
bit 7    underscore
bit 6    reverse video
bit 4    background transparency
         0 = b/g opaque
         1 = b/g transparent
bit 1-0  character set 0,1,2 or 3
```

- Spare Attribute byte (4 byte cells only)

```
for application use
```

CharRect(p32_PS, p32_CharRect, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_CharRect;
        ULONG*  p32_PS;
};
```

This writes a rectangle of alphanumeric characters to the referenced device context. The set of characters and attributes for the rectangle is taken from the presentation space cell buffer. The characters are drawn and clipped according to the window's cell buffer origin, the location of the rectangle relative to that origin, and the size of the window.

p32_PS points to the Vio presentation space.

p32_CharRect
points to a block of parameters for the call.

The parameter block for the call will contain the following:

u32_StartRow
the starting row

u32_StartCol
the starting column in the presentation space of the character rectangle to be output.

u32_RectWidth
the width of the rectangle to be updated.

u32_RectHeight
the height of the rectangle to be updated.

Returns: *short int*

0 ok
!=0 return code

Note: This call will be used to implement the advanced Vio function VioSetOrg. If the origin is moved such that the window background is "exposed" either on the right or at the bottom then the graphics engine must clear the old alphanumeric data from that area of the window.

CharStr(p32_PS, p32_CharStr, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_CharStr;
        ULONG*  p32_PS;
};
```

This writes a string of alphanumeric characters to the referenced device context. The set of characters and attributes for the string is taken from the presentation space cell buffer. The characters are drawn and clipped according to the window's cell buffer origin, the location of the string relative to that origin, and the size of the window. The string will fold at the end of a row and will continue in row-major order either for the given string length or until the Logical Video Buffer is exhausted.

p32_PS points to the Vio presentation space.

p32_CharStr
points to a block of parameters for the call.

The parameter block for the call will contain the following:

u32_StartRow
: the starting row

u32_StartCol
: the starting column in the presentation space of the character string to be output.

u32_StrLength
: the length of the character string to be output.

Returns: *short int*

0 ok
!=0 return code

ScrollRect(p32_PS, p32_ScrollRect, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_ScrollRect;
        ULONG*  p32_PS;
};
```

This function has been included to allow for device drivers that have the capability of BitBlting pels from one region of a window to another. Such a device would only need to update from the presentation space for data outside the window but within the scroll rectangle. A device driver without BitBlt would update completely from the presentation space by suitable adjustment to the rectangle parameters and performing a CharRect function.

p32_PS points to the Vio presentation space.

p32_ScrollRect
: points to a block of parameters for the call.

The parameter block for the call will contain the following:

u32_StartRow
: the starting row

u32_StartCol
: the starting column in the presentation space of the character string to be output.

u32_RectWidth
: the width of the scroll rectangle.

u32_RectHeight
: the height of the scroll rectangle.

u32_HorizCount
: the number of rows to be scrolled (see below).

u32_VertCount
: the number of columns to be scrolled. These two fields define the amount and direction of the scrolling to be done. Positive values define movements downwards and to the right. Negative values define movements upwards and to the left. Currently this function is used to implement VioScrollnn (where nn = Dn, Lf, Rt, Up) and hence for all calls one of the counts will always be zero.

p32_FillCell
: points to a cell (character and attributes) to be used for filling the tail of the scroll region. This cell is only of use when a device driver has used BitBlt. If p32_FillCell is null, the fill will be taken from cells in the Logical Video Buffer.

Returns: *short int*

0 ok
!=0 return code

UpdatePointer(p32_PS, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_PS;
};
```

Sets the alphanumeric cursor position, shape and visibility.

p32_PS points to the Vio presentation space.

Returns: *short int*

0 ok
!=0 return code

This function updates the drawn alphanumeric cursor to match the cursor state information contained in the presentation space. This will usually involve removing the previous cursor from the window and then drawing the new cursor, if visible, according to the presentation space information. The new cursor, if visible, will be positioned and clipped according to this information and the window's cell buffer origin and size.

The cursor is drawn as an XOR bar. It's position, size and shape will be saved by the graphics engine in a reserved area in the Vio presentation space.

There is only one cursor visible on a screen at any one time and this will be in the window that has input focus. The User box must alter the visibility of the cursor when changing input focus.

Collisions are handled below graphics engine interface. The device driver will remove and redraw the cursor if necessary, although a BitBlt operation will copy everything including the cursor.

16.1.12.15 Bitmap Functions

Short list of bitmap function calls:

- BitBlt
- CreateBitmap
- DeleteBitmap
- DeviceCreateBitmap
- DeviceDeleteBitmap
- ScanLR
- PolyScanLine
- GetBitmapBits
- GetBitmapDimension
- GetBitmapParameters
- GetPel
- SaveBits
- SelectBitmap
- DeviceSelectBitmap
- SetBitmapBits
- SetBitmapDimension
- SetPel

BitBlt(u32_SrcDcH, u32_Count, p32_Parm, u32_Mix, u32_Style, u32_TargDcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_TargDcH;
        ULONG   u32_Style;
        ULONG   u32_Mix;
        ULONG*  p32_Parm;
        ULONG   u32_Count;
        ULONG   u32_SrcDcH;
};
```

This copies a rectangle of Bitmap image data from the specified

source Device Context to a target Device Context.

Both source and target may refer to the same Device Context. If this is the case, the copy will be non-destructive if the source and target rectangles overlap.

The Device Contexts may be either memory Device Contexts (with a selected bitmap), or Device Contexts for devices which support raster operations.

The current pattern foreground and background bitmap colors of the target Device Context are used. Also, if the mix requires both source and pattern then a 3-way operation is performed (using the pattern defined by the current pattern of the target DcH) otherwise a 2-way operation is performed. Note that for a *StretchBlt* operation, only the source data and *NOT* the pattern is stretched.

If any of the source data is not available, for example if the source Device Context is connected to a screen window, and the source rectangle is not currently all visible, no error code is returned and the operation proceeds by reading what is there.

Note: Rectangles defined by BitBlt are non-inclusive. They include the left and lower boundaries of the rectangles in device units, but not the right and upper boundaries. Thus if the bottom left maps to the same device pixel as the top right, that rectangle is deemed to be empty.

The color attribute values are used in converting between monochrome and color data. This is the only format conversion performed by BitBlt. The conversions are as follows:

- Outputing a monochrome pattern to a color device

  In this case the pattern is interpreted in two colors, using the current pattern colors:

  — pattern 1s are interpreted as pattern foreground color

  — pattern 0s are interpreted as pattern background color

  The color pattern is then copied to the target without regard for the target's original values.

- BltBlting from a monochrome bitmap to a color bitmap (or device)

  The source bits are converted as follows:

  — source 1s -> image foreground color

  — source 0s -> image background color

- BltBlting from a color bitmap to a monochrome bitmap (or device)

— pels which are the source image background color -> image background color

— all other pels -> image foreground color

Note that in all of these cases it is the attributes of the *target* presentation space which are used.

Parameters:

u32_SrcDcH
Specifies the handle of the source Device Context.

u32_Count
Specifies the number of X,Y pairs of coordinates in the parameter block. *Note:* For Style = 0 and Mix value specifying PatternBlt only, then count must be >= 2, otherwise, for style = 0 (BitBlt) count must be >= 3, otherwise for style 1, 2 or 3 (StretchBlt) count must be >= 4.

p32_Parm
Provides a long pointer to the parameter block:

u32_TargX1, u32_TargY1, u32_TargX2, u32_TargY2,

u32_SrcX1, u32_SrcY1, u32_SrcX2, u32_SrcY2
Specify the bottom left and top right corners respectively of the target and source rectangles respectively in WCS.

Note that the exact number of parameters expected will depend on the setting of u32_Count.

u32_Mix
Specifies a 32 bit raster operation code representing a mix value in the range 0..'FF'X. Each plane of the target can be considered to be processed separately. For any pel in a target plane, three bits together with the Device Context bitmap mix value are used to determine its final value. These are the value of that pel in the pattern (P) and Source (S) data and the initial value of that pel in the Target (T) data. For any combination of P S T pel values, the final target value for the pel is determined by the appropriate Mix bit value as shown in the table below:

```
P  S  T(initial)   T(final)
0  0  0           Mix bit 0 (LS)
0  0  1           Mix bit 1
0  1  0           Mix bit 2
0  1  1           Mix bit 3
1  0  0           Mix bit 4
```

```
1  0  1        Mix bit 5
1  1  0        Mix bit 6
1  1  1        Mix bit 7 (MS)
```

Raster operation code values are as defined for Microsoft Windows.

u32_Style
: Specifies how eliminated lines/columns are treated if a compression is performed.

  0 — Do not stretch or compress the data.

  1 — Stretch/Compress as necessary, OR'ing any eliminated rows/columns. This is used for white on black.

  2 — Stretch/compress as necessary, AND'ing any eliminated rows/columns. This is used for black on white.

  3 — Stretch/Compress as necessary, ignoring any eliminated rows/columns. This is used for color.

  Note: The values 1 to 32K are reserved by the application and values greater than 32K will be passed directly to the device driver. This allows applications to use values of their own for use with "intelligent" devices.

u32_TargDcH
: Specifies the handle of the target Device Context.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

CreateBitmap(p32_Bits, p32_Info, u32_width, u32_Height, u32_Planes, u32_Bitcount, u32_Usage, u32_DcH, u32_FuncNo)

This is used to create a Bitmap of the specified form and to obtain its handle.

Bitmap size is limited by available memory, the maximum width and height are 64K. *Note:* There are several standard bitmap formats which should normally be adhered to. These are:

Bitcount	Planes
1	1
4	1
8	1
24	1

Note: The DC Handle supplied to this function must never be

NULL. The bitmap will be created on the device specified. The bitmap can be selected to a different device later and the Engine will handle transfer of bits from one device to the other but bitmaps always belong to some device.

Note: If the value specified for u32_Planes or u32_Bitcount is incompatible with the physical device specified by u32_DcH then an error will be returned by the engine.

u32_Width, u32_Height
: These integers define the width and height of the Bitmap in pels.

u32_planes
: The number of color planes in the bitmap. For reference: each plane has ((Width*Bitcount+31)/32*4*Height) bytes.

u32_Bitcount
: The number of adjacent color bits per pel.

u32_Usage
: Provides additional information to help the device when creating a new bitmap. The usage information will be interpreted as follows:

  Bit 0 This bit is reserved.

  Bit 1 Discardability

  - '0'B May discard this bitmap if short of storage and it is not currently selected into a DcH or selected as the current pattern (if it has an lcid assigned but is not selected it can be discarded; bitmaps that are to be used as patterns should not be made discardable by the application).
  - '1'B May not discard this bitmap.

  Bit 2 Specifies whether or not p32_Bits and p32_Info are to be used to initialize the newly created bitmap.

  - '0'B Bitmap initialization is device dependent.
  - '1'B Use p32_Bits and p32_Info to initialize the bitmap.

  Note: Bits 16-31 may be used for special reasons known to be supported by the particular device driver.

p32_Bits
: The address in application storage from which the bitmap data is to be copied.

  It is assumed that enough data is supplied to initialize the whole bitmap.

p32_Info
: The address in application storage of the Bitmap Info Table, which describes the format and colors of the data bits.

u32_DcH
: Specifies the handle for the device context. In other words, this specifies the physical device to create the bitmap on or the physical device with which the bitmap must be compatible. A null handle is invalid.

Returns: *HBITMAP*

```
 0 error
!=0 the handle of the created Bitmap
```

DeleteBitmap(u32_BmapH, u32_FuncNo)

This is used to destroy a specified bitmap. This will produce an error if the specified bitmap is currently selected.

Parameters:

u32_BmapH
: The handle of the bitmap to be deleted.

Returns: *BOOL*

```
 0 error
 1 ok
```

DeviceCreateBitmap(p32_Handle, u32_Usage, p32_Address, p32_Info, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Parm;
        ULONG   u32_Usage;
        ULONG*  p32_Handle;
};
```

DeviceDeleteBitmap(p32_Handle, u32_Usage, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Usage;
        ULONG*  p32_Handle;
};
```

ScanLR (p32_SearchData, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_SearchData;
};
```

Scans left or right from the given (x,y) location looking for a pel which satisfies the search condition. This is a private Engine interface.

P32_SEARCHDATA
: Pointer to the search structure defined as follows:

  s32_StartX
  : Search starting X coordinate

  s32_StartY
  : Search starting Y coordinate

  s32_HitX
  : X coordinate of match

  s32_HitY
  : Y coordinate of match

  u32_Color
  : Index of color

  s32_Control
  : Search Control

    D0 = 0, Search for not color
    D0 = 1, Search for color
    D1 = 0, step right
    D1 = 1, step left

PolyScanLine (p32_PSL1, p32_PSL2, p32_BoundingRect, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_BoundingRect;
        ULONG*  p32_PSL2;
        ULONG*  p32_PSL1;
};
```

Fills an area lying between two polyshortlines.

This call is used from the engine to the DDI. The function must be supported by all device drivers.

The device driver can make the following assumptions:

- The two polyshortlines do not cross
- Both polyshortlines have the same s16_StartY and s16_EndY
- For both polyshortlines, if s16_StartY < s16_EndY
  - Every step will be in one of the directions 0, 1, 2, 3, or 7
- For both polyshortlines, if s16_StartY > s16_EndY
  - Every step will be in one of the directions 3, 4, 5, 6, or 7

Whenever direction = 3 or 7 (i.e., horizontal), the pixels defined are outside the area fill, and should not be filled. Thus a device driver should always look ahead to the next non-horizontal step, adjusting current position in X if required, before filling.

The driver should ignore bit 3 (Draw/Skip) of steps.

No clipping is necessary on this figure.

Parameters:

p32_PSL1, PSL2
: Long pointers to the two polyshortlines. These are each POLYSHORT structures, as described for PolyShortLine.

p32_BoundingRect
: This is a rectangle which bounds the whole figure.

GetBitmapBits(u32_ScanStart, u32_ScanCount, p32_Bits, p32_Info, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Info;
        ULONG*  p32_Bits;
        ULONG   u32_ScanCount;
        ULONG   u32_ScanStart;
};
```

This transfers bitmap data from the specified Device Context to application storage. The Device Context must be a memory Device Context, with a bitmap currently selected.

The Bitmap Info Table must be initialized with the values of *cPlanes* and *cBitCount*, for the format of data which it wants. This must be one of the standard formats. On return, *cx*, *cy*, and the *argbColor* array will have been filled in by the system.

Conversion of the bitmap data will have been carried out if necessary.

The *address* must point to a storage area large enough to contain data for the requested number of scanlines. The amount of storage required for one scanline can be determined by calling GetBitmapParameters. It is

```
((bitcount*width + 31)/32)*height*planes*4   bytes
```

There are four standard bitmap formats. All device drivers are required to be able to translate between any of these formats and their own internal formats. The standard formats are as follows:

Bitcount	Planes
1	1
4	1
8	1
24	1

These formats are chosen because they are identical or similar to all formats commonly used by raster devices. Only single plane formats are standard, but it is very easy to convert these to any multiple plane format used internally by a device.

The pixel data is stored in the bitmap in the order of the coordinates as they would appear on a display screen. That is, the pixel in the lower left corner is the first in the bitmap. Pixels are scanned to the right and up from there. The first pixel's bits are

stored beginning in the most significant bits of the first byte. The data for pixels in each scan line is packed together tightly. Each scanline, however, will be padded at the end so that each scan line begins on a ULONG boundary.

Bitmap Color Tables

Each standard format bitmap must be accompanied by a Bitmap Info Table. Because the standard format bitmaps are intended to be traded between devices, the color indices in the bitmap are meaningless without more information. A bitmap info table has the following structure:

```
struct BitmapInfoTable {
  USHORT cx;          /* length of a scanline                    */
  USHORT cy;          /* number of scanlines                     */
  USHORT cPlanes;     /* number of planes (1 if standard         */
                      /* format)                                 */
  USHORT cBitCount;   /* number of bits per pixel                */
  RGB    argbColor[]; /* color table                             */
};
```

The argbColor array is a packed array of 24 bit RGB values. If there are N bits per pixel, then the argbColor array would contain 2^N RGB values, unless N = 24. The standard format bitmap with 24 bits per pixel is assumed to contain RGB values and does not need the argbColor array.

Bitmap Example

To make the ordering of all the bytes clear, consider the following simple example of a 5 x 3 array of colored pixels:

```
              Red   Green Blue  Red   Green
              Blue  Red   Green Blue  Red
              Green Blue  Red   Green Blue

     ExampleBitmap   =
   0x23,0x12,0x30,0x00          /* bottom line */
   0x31,0x23,0x10,0x00          /* middle line */
   0x12,0x31,0x20,0x00          /* top line    */

#define BLACK  0x000000L
#define RED    0x0000FFL
#define GREEN  0x00FF00L
#define BLUE   0xFF0000L

struct BitmapInfoTable ExampleInfo = {
       5,                               /* width */
       3,                               /* height */
       1,                               /* planes */
       4,                               /* bitcount */
       BLACK,RED,GREEN,BLUE,            /* color table */
       BLACK,BLACK,BLACK,BLACK,
       BLACK,BLACK,BLACK,BLACK,
       BLACK,BLACK,BLACK,BLACK
};
```

Parameters:

u32_ScanStart
: The scan-line number at which the data transfer is to start.

u32_ScanCount
: The number of scan lines to be returned.

p32_Bits
: The address in application storage into which the bitmap data is copied.

p32_Info
: The address in application storage of a Bitmap Info Table as described above.

Returns: *long int*

```
-1 Error
>=0 Number of scanlines actually returned
```

GetBitmapDimension(u32_BmapH, p32_XY, u32_FuncNo)

This is used to read back a previously associated width and height from the specified bitmap, in 0.1 mm units. This value can be associated by the SetBitmapDimension call and is not used by the engine.

Parameters:

u32_BmapH
: Specifies the handle of the bitmap whose width and height is required.

p32_XY
: Provides a far pointer to the width and height parameters required as 0.1 mm units.

Returns: *BOOL*

```
0 error
1 ok
```

GetBitmapParameters(u32BmapH, p32_Parm, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
                                        USHORT* p32_Parm;
        ULONG   u32_BmapH;
};
```

Returns information about the bitmap identified by the specified bitmap handle.

Parameters:

u32_BmapH
: The handle of the bitmap.

p32_Parm
: Provides a long pointer to a data area in which the returned data about the specified bitmap is stored. The structure is the first four elements (width, height, planes, bitcount) of a Bitmap Info Table (see GetBitmapBits).

Returns: *BOOL*

0 error
1 ok

GetPel(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

This function gets a pel from a position specified in WCS (world coordinates).

Parameters:

p32_xy Provides a long pointer to the coordinate pair (s32_x, s32_y).

Returns: *long int*

-1 error
>=0 color index value for the pel

SaveBits(p32_rect, u32_Options, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Options;
        ULONG   p32_rect;
};
```

This function copies bits from the screen to a bitmap managed by the device driver, and also for the screen to be subsequently restored from this bitmap.

It is used by the user interface routines (not the API) to improve the performance of dialog boxes.

This function is not a required function, but the driver must return 'failure' if it does not do it.

Parameters:

p32_Rect
: points to a screen rectangle

u32_Options
: Command flags values are:

```
0 - save a bitmap
1 - restore the bitmap
2 - discard the bitmap without using it
```

Returns: *BOOL*

```
0 error
1 ok
```

SelectBitmap(u32_BmapH, u32_DcH, u32_FuncNo)

This is used to select a specified bitmap into a selected Memory Device Context or, if called with a null bitmap handle to deselect the specified bitmap. The handle of the previously selected bitmap is returned.

It is an error if the specified bitmap is already selected into any DC.

If the specified bitmap and device are incompatible then an error will be returned by the engine.

Parameters:

u32_BmapH
: The handle of the bitmap to be selected.

Returns: *HBITMAP*

```
  -1 error
   0 null handle
>0||<-1 handle of the previously selected bitmap
```

DeviceSelectBitmap (Handle, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Handle;
};
```

SetBitmapBits(u32_ScanStart, u32_ScanCount, p32_Bits, p32_Info, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Info;
        ULONG*  p32_Bits;
        ULONG   u32_ScanCount;
        ULONG   u32_ScanStart;
};
```

This transfers bitmap data from application storage into the specified Device Context.

The Device Context must be a memory Device Context, with a bitmap currently selected. Note that this function will not set bits directly to any other kind of device.

If the format of the supplied bitmap does not match that of the device, it is converted, using the supplied Bitmap Info Table. Only the standard formats will be supported.

Parameters:

u32_ScanStart
: The scan-line number at which the data transfer is to start.

u32_ScanCount
: The number of scan lines to be transmitted.

p32_Bits
: The address in application storage from which the bitmap data is to be copied.

P32_Info
: The address in application storage of the Bitmap Info Table.

Returns: *long int*

```
-1 Error
>=0 Number of scanlines actually set
```

SetBitmapDimension(u32_BmapH, p32_Parm, u32_FuncNo)

This is used to associate a width and a height with the specified bitmap in 0.1 mm units. This value can be read back by the GetBitmapDimension call and is not used by the application.

Parameters:

u32_BmapH
: Specifies the handle of the bitmap to be used.

p32_Parm
: Specifies a pointer to the parameters:

  u32_Width, u32_Height
  : The width and height in 0.1 mm units to be associated with the bitmap.

Returns: *BOOL*

0 error
1 ok

SetPel(p32_Parm, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Parm;
};
```

This function sets a pel at a position specified in WCS to the current line attribute color and mix.

Parameters:

p32_Parm
: Provides a long pointer to a parameter block:

  u32_X, u32_Y
  : Specifying a position in WCS.

Returns: *short int*

0 error
1 ok
2 CorrelateHit

16.1.12.16 Region Functions

Short list of the function calls:

- CombineRegion
- ComputeRegions
- CreateRectRegion
- DestroyRegion
- EqualRegion

- ExcludeClipRectangle
- GetClipBox
- GetClipRects
- GetRegionBox
- GetRegionRects
- IntersectClipRectangle
- OffsetClipRegion
- OffsetRegion
- PaintRegion
- PtInRegion
- PtVisible
- QueryClipRegion
- QueryVisRegion
- RectInRegion
- RectVisible
- SelectClipRegion
- SelectVisRegion
- SetRectRegion
- NotifyClipChange

CombineRegion(u32_DestRgnH, u32_Src1RgnH, u32_Src2RgnH, u32_Mode, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Mode;
        ULONG   u32_Src2RegnH;
        ULONG   u32_Src1RgnH;
        ULONG   u32_DestRrnH;
};
```

This combines two regions to make a third.

Parameters:

u32_DestRgnH
: The handle of the destination region.

u32_Src1RgnH, u32_Src2RgnH
: The handles of the two regions to be combined.

u32_Mode

Method of combination, as follows:

```
1 OR   - Union of Src1 and Src2
2 COPY - Src1 only (Src2 ignored)
4 XOR  - Symmetric difference of Src1 and Src2
6 AND  - Intersection of Src1 and Src2
7 DIFF - Src1 and not(Src2)
```

Returns: *short int*

```
0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)
```

ComputeRegions(u32_DcH u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This forces computation of the Rao region. (Note that the Rao region is the region reflecting the visible area on the screen). Note that SetDCOrg does not force recompute.

Returns: *short int*

```
0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)
```

CreateRectRegion(p32_xy, u32_count, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Count;
};
```

This creates a region defined using a series of rectangles. The new region is defined by the OR of all the rectangles.

If u32_count is zero, an empty region is created.

Parameters:

p32_xy A far pointer to the region definition which is an array of x,y pairs in device coordinates. Odd x,y pairs specify the minimum coordinates of a rectangle and even x,y pairs specify the maximum coordinates of a rectangle.

u32_count
: A count of the number of rectangles in the region definition.

Returns: *HRGN*

0 error
!=0 region handle

DestroyRegion(u32_RgnH, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_RgnH;
};
```

This destroys the specified region unless it has been selected as a clipping region.

Parameters:

u32_RgnH
: The handle of the region.

Returns: *BOOL*

0 error
1 ok

EqualRegion(u32_Src1RgnH, u32_Src2RgnH, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Src2RgnH;
        ULONG   u32_Src1RgnH;
};
```

This checks whether two regions are identical.

Parameters:

u32_Src1RgnH, u32_Src2RgnH
: The handles of the two regions to be checked.

Returns: *short int*

0 error
1 not equal
2 equal

ExcludeClipRectangle(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

Excludes the specified rectangle from the clipping region. The bottom and left boundaries of the rectangle are included in the boundary which is to disappear.

Parameters:

p32_xy A far pointer to an array s32_x1, s32_y1, s32_x2, s32_y2 where s32_x1, s32_y1 specifies the minimum coordinates of the rectangle and s32_x2, s32_y2 specifies the maximum coordinates of the rectangle in world coordinates.

Returns: *short int*

0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)

GetClipBox(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

Returns the dimensions of the tightest rectangle around the DC region. The DC region is the intersection of the visible region, clip region, viewing limits, graphics field and clip area.

Parameters:

p32_xy A far pointer to an array s32_x1, s32_y1, s32_x2, s32_y2 in which the rectangle is returned where s32_x1, s32_y1 returns the minimum coordinates of the rectangle and s32_x2, s32_y2 returns the maximum coordinates of the rectangle in world coordinates.

Returns: *short int*

0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)

GetClipRects(p32_BoundRect, p32_Control, p32_xy, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_xy
    ULONG* p32_Control
    ULONG* p32_BoundRect
};
```

This returns a list of x,y coordinate pairs specifying the clip region associated with the specified DC.

Returns the list of x,y coordinate pairs for rectangles specifying the region and intersecting an optional bounding rectangle. By updating the start rectangle number value, the function can be called multiple times to allow for more rectangles than can be stored in the receiving buffer.

Parameters:

p32_BoundRect
: a far pointer to a bounding rectangle. The first x,y pair define the minimum coordinates of the rectangle and the second x,y pair define the maximum coordinates of the rectangle in device coordinates. Only rectangles intersecting this bounding rectangle will be returned. If this pointer is NULL, all rectangles in the region will be enumerated. If p32_BoundRect is not NULL, then the each of the rectangles returned in p32_XY will be the intersection of the bounding rectangle with a rectangle in the region.

  If p32_BoundRect is not NULL, then each of the rectangles returned in p32_xy will be the intersection of the bounding rectangle with a rectangle in the region.

p32_Control
: A far pointer to a structure containing the following elements.

  u16_Start
  : The rectangle number to start enumerating at. A 0 value means the same as 1; i.e., start at the beginning.

  u16_Bufsize
  : The number of rectangles that will fit into the buffer. A value of at least 1 is supplied.

  u16_Num_Written
  : A returned value indicating how many rectangles were written into the buffer. A value

below u16_Start means that there are no more rectangles to enumerate.

u16_Direction
The direction the rectangles are listed.

1 => left to right, top to bottom

2 => right to left, top to bottom

3 => left to right, bottom to top

4 => right to left, bottom to top

p32_xy A far pointer to a region definition which is an array of x,y pairs in device coordinates. Odd x,y pairs specify the minimum coordinates of a rectangle and even x,y pairs specify the maximum coordinates of a rectangle. The format is identical to that for CreateRectRegion.

Returns: *short int*

```
0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)
```

GetRegionBox(u32_RgnH, p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
        ULONG   u32_RgnH;
};
```

Returns the dimensions of the tightest rectangle around a region. If the region is empty, the rectangle returned will have the right boundary less than the left and the top boundary less than the top.

A null region is where the boundaries left = right and/or bottom = top.

Parameters:

u32_RgnH
The handle of the region.

p32_xy A far pointer to an array s32_x1, s32_y1, s32_x2, s32_y2 in which the rectangle is returned where s32_x1, s32_y1 returns the minimum coordinates of the rectangle and s32_x2, s32_y2 returns the maximum coordinates of the rectangle in device coordinates.

Returns: *short int*

```
0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)
```

GetRegionRects(u32_RgnH, p32_BoundRect, p32_Control, p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
        ULONG*  p32_Control;
        ULONG*  p32_BoundRect;
        ULONG   u32_RgnH;
};
```

This returns a list of x,y coordinate pairs specifying the region associated with the given region handle. A region selected as a clipping region can also be specified.

Parameters:

u32_RgnH
: The region handle specifying which region data to be returned.

p32_Control
: A far pointer to a structure containing the following elements.

  u16_Start
  : The rectangle number to start enumerating at. A value of 0 means start at the beginning, a value of 1 means start at the second, etc.

  u16_Bufsize
  : The number of rectangles that will fit into the buffer. A value of at least 1 is supplied.

  u16_Num_Written
  : A returned value indicating how many rectangles were written into the buffer. A value below u16_bufsize means that there are no more rectangles to enumerate.

  u16_Direction
  : The direction the rectangles are listed.

    1 => left to right, top to bottom

    2 => right to left, top to bottom

3 => left to right, bottom to top

4 => right to left, bottom to top

p32_BoundRect
: a far pointer to a bounding rectangle. The first x,y pair define the minimum coordinates of the rectangle and the second x,y pair define the maximum coordinates of the rectangle in device coordinates. Only rectangles intersecting this bounding rectangle will be returned. If this pointer is NULL, all rectangles in the region will be enumerated.

  If p32_BoundRect is not NULL, then each of the rectangles returned in p32_xy will be the intersection of the bounding rectangle with a rectangle in the region.

p32_xy
: A far pointer to a region definition which is an array of x,y pairs in device coordinates. Odd x,y pairs specify the minimum coordinates of a rectangle and even x,y pairs specify the maximum coordinates of a rectangle. The format is identical to that for CreateRectRegion.

Returns: *BOOL*

0 error
1 ok

IntersectClipRectangle(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

Sets the new clipping region to the intersection of the current clip region and the specified rectangle.

Parameters:

p32_xy
: A far pointer to an array s32_x1, s32_y1, s32_x2, s32_y2 where s32_x1, s32_y1 specifies the minimum coordinates of the rectangle and s32_x2, s32_y2 specifies the maximum coordinates of the rectangle in world coordinates.

Returns: *short int*

0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)

OffsetClipRegion(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

Moves the clipping region by the specified amounts.

Parameters:

p32_xy The s32_x, s32_y offsets by which the clipping region is to be moved in in world coordinates.

Returns: *short int*

0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)

OffsetRegion(u32_RgnH, p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_RgnH;
};
```

This moves the given region by the specified offsets unless the region is in use as a clipping region.

Parameters:

u32_RgnH
The handle of the region to be moved.

p32_xy The s32_x, s32_y offsets by which the region is to be moved in device coordinates.

Returns: *BOOL*

0 error
1 ok

PaintRegion(u32_RgnH, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_RgnH;
};
```

This function paints the specified region using the current pattern attributes.

Parameters:

u32_RgnH
: The handle of the region.

Returns: *short int*

0 error
1 ok
2 CorrelareHit

PtInRegion(u32_RgnH, p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
        ULONG   u32_RgnH;
};
```

This checks whether a point lies within a region.

Parameters:

u32_RgnH
: The handle of the region.

p32_xy Specifies the s32_x, s32_y point in device coordinates.

Returns: *short int*

0 error
1 not in region
2 in region

PtVisible(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

This checks whether a point is visible within the clipping region of the specified device context (where clipping region is defined to be the intersection of application clipping and clipping resulting from windowing).

Parameters:

p32_xy Specifies the s32_x, s32_y point in world coordinates.

Returns: *short int*

```
0 error
1 not visible
2 visible
```

QueryClipRegion(u32_DcH u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This returns the handle of the currently selected clip region. If there is no currently-selected clip region, NULL is returned.

Returns: *HRGN*

```
-1 error
 0 null handle (no region selected)
<-1 region handle
 >0 region handle
```

QueryVisRegion(u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

Returns the handle of the current visible region.

Returns: *HRGN*

```
-1 error
 0 null handle (no region selected)
<-1 region handle
 >0 region handle
```

RectInRegion(u32_RgnH, p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
        ULONG   u32_RgnH;
};
```

This checks whether any part of a rectangle defined by the specified coordinates lies within the the specified region.

Parameters:

u32_RgnH
: The handle of the region.

p32_xy A far pointer to an array s32_x1, s32_y1, s32_x2, s32_y2 where s32_x1, s32_y1 specifies the minimum coordinates of the rectangle and s32_x2, s32_y2 specifies the maximum coordinates of the rectangle in device coordinates.

Returns: *short int*

```
0 error
1 not in region
2 partially in region
3 all in region
```

RectVisible(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

This checks whether any part of the bounding rectangle defined by the specified coordinates is visible within the clipping region of the specified device context (where clipping region is defined to be intersection of application clipping and clipping resulting from windowing).

Parameters:

p32_xy A far pointer to an array s32_x1, s32_y1, s32_x2, s32_y2 where s32_x1, s32_y1 specifies the minimum coordinates of the rectangle and s32_x2, s32_y2 specifies the maximum coordinates of the rectangle in world coordinates.

Returns: *short int*

```
0 error
1 not visible
2 partially visible
3 all visible
```

SelectClipRegion(u32_RgnH, p32_OldRgnH, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_OldRgnH;
        ULONG   u32_RgnH;
};
```

Specifies the region to be used for clipping, when any drawing takes place in the specified device context. The handle of the

previous selected region is returned. A null returned handle means that the default clip region was in use before the select.

A region can only be selected by one DC at any one time and when selected region operations modifying the region are invalid.

The coordinates of the region are taken to be device coordinates within the device context.

Clipping is inclusive at the left and bottom boundaries and exclusive at the right and top boundaries.

Functions that modify the clipping region also modify the region when it's handle is returned after another SelectClipRegion.

Parameters:

u32_RgnH
: The handle of the region. If it is null, the clipping region is set to no clipping, its initial state.

p32_OldRgnH
: The handle of the previously selected region. A null handle means that there was no clipping.

Returns: *short int*

```
0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)
```

SelectVisRegion(u32_RgnH, p32_OldRgnH, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_OldRgnH;
        ULONG   u32_RgnH;
};
```

Specifies the region to be used for clipping, when any drawing takes place in the specified device context. A vis region is used to define the visible portion of a Window on the screen. The vis region will be combined with the clip region if present to form the DC region.

The handle of the previous selected vis region is returned. A null returned handle means that the default vis region was in use before the select.

A region can only be selected by one DC at any one time and when selected region operations modifying the region are invalid.

The coordinates of the region are taken to be device coordinates within the device context.

Clipping is inclusive at the left and bottom boundaries and exclusive at the right and top boundaries.

Parameters:

u32_RgnH
: The handle of the region. If it is null, the clipping region is set to no clipping, its initial state.

p32_OldRgnH
: The handle of the previously selected region. A null handle means that there was no vis region selected.

Returns: *short int*

0 error
1 NULL region
2 RECTangular region
3 COMPLEX region (more than 1 rectangle)

SetRectRegion(u32_RgnH, p32_xy, u32_count, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Count;
        ULONG*  p32_xy;
        ULONG   u32_RgnH;
};
```

Sets the specified region to the specified region definition given by a series of rectangles unless the region is in use as a clipping region. The region is defined by the OR of all the rectangles.

If u32_count is zero, an empty region is created.

Parameters:

u32_RgnH
: The handle of the region.

p32_xy
: A far pointer to the region definition which is an array of x,y pairs in device coordinates. Odd x,y pairs specify the minimum coordinates of a rectangle and even x,y pairs specify the maximum coordinates of a rectangle. The series of rectangles so defined specify the new region data.

u32_count
: A count of the number of rectangles in the region definition.

Returns: *BOOL*

0 error
1 ok

NotifyClipChange (p32_Rect, u32_Complexity, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Complexity;
        ULONG*  p32_Rect;
};
```

This function is called whenever the clip region intersected with the visible region is changed. This function is not required. It can be handled completely with a far return if the device driver is not interested in each clip region change.

Parameters:

U32_COMPLEXITY
: Indicates the number of rectangles in the new clip region.

P32_RECT
: A far pointer to a rectangle which bounds the new region. If the region is a single rectangle, this will be the same rectangle.

Returns: *TBD*

16.1.12.17 Font Functions

There are a number of base character sets (at least 6) loaded at system initialization by Presentation Manager. The engine is responsible for the loading of base pattern and marker sets. The method of referencing base pattern and marker sets is described in the attributes section of this document. The engine is also responsible for the loading of any base fonts.

Short list of the function calls:

- CreateLogicalFont
- QueryLogicalFont
- DeleteSetId
- EnableKerning
- GetCodePage
- GetKerningPairTable
- QueryWidthTable
- RealizeFont
- GetDCCaps

- LoadFont
- QueryBitmapHandle
- QueryFontAttributes
- QueryFonts
- DeviceQueryFontAttributes
- DeviceSetAVIOFont
- DeviceQueryFonts
- QueryKerning
- QueryNumberSetIds
- QuerySetIds
- SetBitmapID
- SetCodePage
- UnloadFont
- GetCodepageTable
- GetRevCodeTable

CreateLogicalFont(u32_LCID, p32_Name, p32_FontMetrics, u32_DcH, u32_FuncNo)

This selects a font from the list of loaded fonts with the closest match in terms of its attributes.

If the specified LCID is already in use an error will be raised.

Font Selection

When GpiCreateLogFont is issued, a physical font is selected which will be used for this logical font. The choice is made in one of two ways:

1. The system examines the logical font attributes which have been specified, and selects which of the physical fonts available to it best matches them, or
2. The application, having already determined (by using GpiQueryFonts) which font it wants, specifies the same *match* value in the logical font attributes as was returned for that font by GpiQueryFonts.

Once the choice has been made, it is never subsequently changed for a particular logical font.

p32_FontMetrics

Field Name	Field Type	Value
Length of Record	2 byte integer	62
Match	4 byte integer	-
Typeface Name	32 byte string	-
Registry ID	2 byte integer	-
Code Page	2 byte integer	-
Height	4 byte integer	-
Average Width	4 byte integer	-
Width Class	2 byte integer	-
Weight Class	2 byte integer	-
Selection Flags	2 byte flags	-
Type Flags	2 byte flags	-
Quality	2 byte integer	-
Font use flags	2 byte integer	-

Notes.

- Match

  QueryFonts will return a unique Match number for each font. CreateLogicalFont can specify Match equal to this number to force selection of a particular font. If Match specifies zero then mapping takes place as normal.

- Height

  The height in world coordinates

- Average Width

  The average width in world coordinates

- Quality

  There are three levels of quality which refer to the perceived quality of the characters printed. These definitions of quality have no exact match in u32_FuncNo.

  - Proof - X'02'

    Perceived quality is paramount, even at the expense of not matching the logical attributes of the font - principally its size.

  - Draft - X'01'

    Perceived quality is less important than matching the logical attributes of the font.

  - Default - X'00'

    The appearance of the text does not matter.

- Font use flags.

```
                                    (bit 0)  - 0 = Reserved

FATTR_FONTUSE_NOMIX                 (bit 1)  - 0 = Normal mixing rules apply
                                               1 = Permissive mixing, as
                                                   defined below
FATTR_FONTUSE_OUTLINE               (bit 2)  - 0 = Font need not be outline
                                                   font
                                               1 = Font must be outline font
FATTR_FONTUSE_TRANSFORMABLE         (bit 3)  - 0 = Font need not be trans-
                                                   formable
                                               1 = Font must be transformable
                                  (bits 4-15)  - 0 = Reserved
```

If FONTUSE_NOMIX is set, it means that when characters are drawn with this font, it does not matter whether the current character mix and background mix attributes are

- honored, or
- temporarily replaced by overpaint and leave-alone, respectively.

Furthermore, if any other primitives are subsequently drawn at the same position, the mixing effect is undefined.

Fonts and Character Attributes

The interaction between fonts and character attributes depends upon whether the transformable flag (FATTR_FONTUSE_TRANSFORMABLE) in the logical font attributes is set.

- If FATTR_FONTUSE_TRANSFORMABLE is set, the lAveCharWidth and lMaxBaselineExt in the font attributes are not currently used and should be zero. When character strings are drawn with this logical font, the sizes of the characters will be determined by the current values of the character attributes. The characters will be positioned, rotated, sheared, etc., precisely as required.

  The transformation is calculated by mapping the box defined by the max_char_increment and em_height in the font metrics to the character box, under the influence of the character angle and shear.

  The value of character mode is ignored when drawing character strings in this case.

- If FATTR_FONTUSE_TRANSFORMABLE is not set, then the lAveCharWidth and lMaxBaselineExt define the size of the font which will be used. This will not be affected by the character box attribute. Either character modes 1 or 2 may be used with such a font:

- In mode (i.e., precision) 1 the start of the string will be positioned precisely (taking into account text alignment), and subsequent characters will be positioned according to the dictates of the font. Character box, angle, shear, extra, break extra, and spacing will all be ignored.
- In mode (i.e., precision) 2, each character will be positioned taking into account all of the character attributes, but the characters themselves will not be scaled, rotated or sheared.

- Note that an attempt to draw a character string in either of the following cases will raise an error:
  - with mode (i.e., precision) = 3 and FATTR_FONTUSE_TRANSFORMABLE not set.
  - with mode (i.e., precision) not = 3 and FATTR_FONTUSE_TRANSFORMABLE set.

Positioning is by the character reference point. This is defined within the font.

u32_LCID
: The LCID which is to be assigned to the font. If the LCID is already assigned, the existing definition will be removed and replaced with the new one. *Note:* lcids -2, -3 and -4, represent AVIO lcids 1, 2 and 3 respectively (it is the function of the Engine to perform the remapping between these values and 1, 2 and 3 for the Device Driver).

p32_Name
: An 8 character name used to describe the logical Font.

Returns: *BOOL*

```
0 error
1 ok
```

QueryLogicalFont(u32_LCID, p32_Name, p32_FontMetrics, u32_DcH, u32_FuncNo)

This returns the font metrics for the logical font loaded onto the specified LCID.

If the specified LCID is in use for a bitmap an error will be raised.

p32_FontMetrics
: A pointer to a Font Metrics structure as specified in CreateLogicalFont.

u32_LCID
: The LCID of the required Logical Font. *Note:* lcids -2,

-3 and -4, represent AVIO lcids 1, 2 and 3 respectively (it is the function of the Engine to perform the remapping between these values and 1, 2 and 3 for the Device Driver).

p32_Name
: A pointer to an 8 character name used to describe the logical Font.

DeleteSetId (u32_LCID, u32_DcH, u32_FuncNo)

This deletes a character set. If the LCID specifies a bitmap id, the bitmap ID is deleted. Base sets cannot be deleted.

An LCID value of 'FFFFFFFF'X (i.e., -1 if this were a signed number), will cause all loaded lcids (i.e., logical Fonts and Bitmap IDs) to be destroyed.

u32_LCID
: Denotes the LCID of the existing character set. *Note:* lcids -2, -3 and -4, represent AVIO lcids 1, 2 and 3 respectively (it is the function of the Engine to perform the remapping between these values and 1, 2 and 3 for the Device Driver).

Returns: *BOOL*

0 error
1 ok

EnableKerning (u32_Flags, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Flags;
};
```

This enables or disables pair kerning. the default is that it is disabled when a DC is created.

u32_Flags
: Denotes whether pair kerning is on or off

```
EK_PAIR (bit 0)     - 0 = pair kerning off
                    - 1 = pair kerning on
```

Returns: *BOOL*

0 error
1 ok

GetCodePage (u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This obtains the current code page.

Returns: *long int*

0 error
!=0 codepage

GetKerningPairTable (u32_count, p32_KernPairs, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_KernPairs;
        ULONG   u32_Count;
};
```

Gets the kerning pairs of the current font.

u32_count
: The number of kern pairs the application wants

p32_KernPairs
: A far pointer to an array of kern pair records. Each record has the form:

```
Word  Char1
Word  Char2
Word  KernAmount

where  Char1        Code point for first character
       Char2        Code point for second character
       KernAmount   2 byte signed integer, indicating the
                    amount of kerning, with positive numbers
                    meaning increase inter-character spacing.
```

Note. The number of kern pairs is a field in the text metrics.

Returns: *BOOL*

0 error
1 ok

QueryWidthTable (u32_sFirstChar, s32_Count, p32_WidthTable, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_WidthTable;
        ULONG   u32_Count;
        ULONG   u32_sFirstChar;
};
```

This returns the width table information for the currently selected logical font in world coordinates.

Parameters:

p32_WidthTable
: A far pointer to a buffer in which the width table data is returned.

u32_Count
: The size in bytes of the buffer pointed to by p32_WidthTable.

u32_sFirstChar
: The codepoint of the initial character for which width table information is required.

Returns: *BOOL*

```
0 error
1 ok
```

RealizeFont (u32_PFont, u32_EFont, u32_Accelator, u32_Command, p32_LogFont, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_LogFont;
        ULONG   u32_Command;
        ULONG   u32_Accelator;
        ULONG   u32_EFont;
        ULONG   u32_PFont;
};
```

This allows the device to attempt to realize a font. The device is called first to realize a font. The engine will then check its tables of generic fonts for a match. The engine then picks the better of the two.

Note: This is a private Engine interface.

The form of the dialog is:

1. The engine asks the driver if it can realize a device font for the log font. The driver answers no by returning zero and yes by returning a 32 bit number for that font and filling in the PFONT data structure.
2. The engine looks for a match amount its generic fonts.
3. The engine chooses the best match between the driver and the generic font.
4. If a generic font has been chosen the driver is asked if it wants to down load it. The driver answers no by returning zero and yes by returning a 32 bit number for that font and filling in the PFONT data structure.

Parameters:

U32_PFont
: if (Command == DeviceFont, LoadEngineFont) The driver fills this in with a 16 bit identifier if the font is realized. Else if (Command == DeleteFont) The driver is given the indentifier it uses for a device font.

p32_EFont
: A long pointer to an engine font supplied by the engine when command == LoadEngineFont.

u32_Accelator
: The device driver sets bits here that tell the engine what the driver would like the engine to perform.

  Possible bits are:

```
TC_BOLD          equ 00001B   ; Wants Embolding
TC_ITALIC        equ 00010B   ; Wants Italicizing
TC_UNDERLINE     equ 00100B   ; Wants Underlining
TC_DUNDERLINE    equ 00100B   ; Wants Double underlining
TC_STRIKEOUT     equ 01000B   ; Wants to be StrikeOut
```

u32_Command
: A 32 bit command which is one of:

  DeviceFont
  : The driver is asked whether it can realize a match.

  LoadGenericFont
  : The driver is asked if it can download an engine generic font.

  DeleteFont
  : The driver is asked to delete a font.

p32_LogFont
A long pointer to a logical font data structure if command == DeviceFont.

GetDCCaps (p32_Flags, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Flags;
};
```

This function is used by the engine to ask the device driver what operations it is capable of, with the present attributes set for the given DC. The device driver is expected to set the flags pointed to by P32_FLAGS as follows:

BIT 1 Set if the device driver can do bounding.

BIT 2 Set if the device driver can do correlations.

BIT 8 Set if device driver can draw lines with the present attributes.

BIT 9 Set if device driver can draw curves with the present attributes.

BIT 10 Set if the device driver can fill areas with the present attributes.

BIT 11 Set if the device driver can draw markers with the present attributes.

All other bits must not be modified. The engine will simulate any operations that the device driver cannot perform.

LoadFont(p32_Filename, u32_FuncNo)

This loads fonts from the specified resource file. All of the fonts in the file become available for the applications to use.

Parameters:

p32_Filename
A long pointer to the filename of the font resource file.

Returns: *BOOL*

0 error
1 ok

QueryBitmapHandle (u32_LCID, u32_DcH, u32_FuncNo)

This returns the bitmap handle for the specified LCID, or raises an error if the LCID does not reference a Bitmap.

u32_LCID

Specifies the LCID for which the bitmap handle is required.

Returns: *HBITMAP*

```
 0 error
!=0 Bitmap handle
```

QueryFontAttributes (u32_AttrsLen, p32_FontAttributes, u32_DcH, u32_FuncNo)

This obtains the attributes of the font currently selected via SetCharSet.

u32_AttrsLen

The length of the font attributes buffer pointed to by p32_FontAttributes.

p32_FontAttributes

The Font File format consists of two sections. The first section contains the general attributes of the font, describing features of the font such as its typeface style and its nominal size. The second section contains the actual definitions of the characters belonging to the font. Each of the sections is described in the following sections.

Note: The names and formats of fields in this section conform to the u32_FuncNo.

```
szFamilyname            32 byte string
szFacename              32 byte string

idRegistry              Word
CodePage                Word

lEmHeight               Word
lXHeight                Word

lMaxAscender            Word
lMaxDescender           Word
lLowerCaseAscent        Word
lLowerCaseDescent       Word

lInternalLeading        Word
lExternalLeading        Word

lAveCharWidth           Word
lMaxCharInc             Word

lMaxBaselineExt         Word
sCharSlope              Word
sInlineDir              Word
sCharRot                Word
```

```
usWeightClass          Word
usWidthClass           Word

sXDeviceRes            Word
sYDeviceRes            Word

sFirstChar             Word
sLastChar              Word
sDefaultChar           Word
sBreakChar             Word

sNominalPointSize      Word
sMinimumPointSize      Word
sMaximumPointSize      Word
fsType                 Word
fsSelection            Word
Capabilities           Word

lSubscriptSize         Word
lSubscriptPosition     Word
lSuperscriptSize       Word
lSuperscriptPosition   Word
UnderscoreWidth        Word
UnderscoreSpacing      Word
lStrikeoutSize         Word
lStrikeoutPosition     Word
KerningPairs           Word
Reserved               Word
Match                  Dword
```

Note: For more detail see Font Attributes section.

Returns: *BOOL*

0 error
1 ok

QueryFonts(p32_FaceName, p32_Metrics , u32_MetricLen, p32_FontCount, u32_DcH, u32_FuncNo)

This returns a record providing details of the fonts, which match the specified *FaceName.*

By inspecting the returned data, the application may choose which of the available fonts is most appropriate for its requirements. If necessary, it can force selection of a particular font.

By specifying a count of zero, the returned value can be used to determine how many fonts match the FaceName.

Note: All sizes are returned in world coordinates.

Parameters:

p32_FaceName
: A far pointer to a null terminated character string specifying the facename.

p32_Metrics
: A far pointer to an array of font element records in which the metrics of matching fonts are returned. The format of each element is as described for QueryFontAttributes. No more than *u32_MetricLen* bytes will be returned for any one font, and the number of fonts returned is limited to the value specified by p32_FontCount.

u32_MetricLen
: The number of bytes of each metrics structure in the p32_Metrics array.

p32_FontCount
: A long pointer to u32_Fontcount which Specifies the number of fonts for which the application wants metrics. On return this is updated with the number of fonts for which metrics are returned.

Returns: *long int*

```
-1 error
>=0 number of fonts not returned (this allows the application
     to determine the number of fonts by specifying count = 0)
```

DeviceQueryFontAttributes(p32_Metrics, u32_iMetrics, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_iMetrics;
        ULONG*  p32_Metrics;
};
```

This obtains the metrics of the currently selected font.

u32_iMetrics
: The size of the data structure pointed to by p32_Metrics.

p32_Metrics
: A long pointer to a font metric block where the information is to be returned.

Returns: *BOOL*

```
 0 error
 1 ok
```

DeviceSetAVIOFont(u32_FontDef, u32_LCIDIndex, u32_DcH, u32_FuncNo)

```
place=inline frame = box
```

```
struct ARGUMENTS {
     ULONG  u32_FuncNo;
     HDC    u32_DcH;
     ULONG  u32_LCIDIndex;
     ULONG* p32_FontDef;
};
```

This call supports loadable cell image sets for AVIO presentation spaces.

The function's result will be true if the given font is acceptable for use with an AVio presentation space and false otherwise. As a side effect the device driver will derive a far pointer to the bit array which constitutes the image data for the given font and cache that address within the DDC corresponding to hDC.

Subsequent CharRect, CharStr, and ScrollRect calls will use those cached addresses in the DDC to materialize character images for LCID 1..3.

u32_LCIDindex
: Specifies the lcid value of 1, 2 or 3.

p32_FontDef
: p32_FontDef is either a far pointer to a Font data structure or a device handle with zero in the HIWORD(FontDef).

Returns: *BOOL*

0 error
1 ok

DeviceQueryFonts(p32_Filter,
p32_Metrics,u32_iMetrics,u32_iFonts,u32_DcH,u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  u32_iFonts;
        ULONG   u32_iMetrics;
        ULONG*  p32_Metrics;
        ULONG*  p32_Filter;
};
```

Enumerates the fonts on a device. For each font on the device the function returns the metrics of the font.

p32_iFonts
: The number of fonts for which the application wants metrics. The number the application actually receives is returned here.

u32_iMetrics
The size of each metrics structure in the array pointed at by p32_Metrics.

p32_Metrics
A long pointer to an array of textmetric records.

p32_Filter
A long pointer to the facename to match.

Returns: *long int*

```
-1 error
>=0 number of fonts not returned (this allows the application
    to determine the number of fonts by specifying count = 0)
```

QueryKerning (p32_Flags, u32_DcH, u32_FuncNo)

This queries the character kerning flags.

p32_Flags
A far pointer to the returned character kerning flags

```
EK_PAIR (bit 0)     - 0 = pair kerning off
                    - 1 = pair kerning on
```

Returns: *BOOL*

```
0 error
1 ok
```

QueryNumberSetIds(u32_LCIDrange, u32_DcH, u32_FuncNo)

Returns the total number of lcids (i.e., logical fonts and Bitmap Ids) which have been loaded.

u32_LCIDrange
Specifies whether GPI Lcids or AVIO lcids or both are to be queried. *Note:* lcids -2, -3 and -4, represent AVIO lcids 1, 2 and 3 respectively (it is the function of the Engine to perform the remapping between these values and 1, 2 and 3 for the Device Driver).

```
1 GPI
2 AVIO
3 GPI & AVIO
```

Returns: *long int*

```
-1 error
>=0 number of lcids
```

QuerySetIds(u32_n, p32_types, p32_names, p32_LCIDs, u32_LCIDrange, u32_DcH, u32_FuncNo)

This queries the first u32_n loaded LCIDs in the Dc. A list of loaded LCIDs, together with their type (Logical Font or Bitmap) are returned. If there are less then u32_n loaded LCIDs, the

remaining types and LCIDs are set to zero.

u32_n Number of LCIDs to query.

p32_types
Pointer to an array of u32_n elements (element type is s32), in which the array elements signify the type of LCID, as follows

```
where   type = 6 logical font
             = 7 bitmap
```

p32_names
A far pointer to an array of pointers to 8-byte fields in which the 8-character names associated with the corresponding LCIDs are returned.

p32_LCIDs
Pointer to an array of u32_n elements (element type is u32) in which the array elements are the queried loaded LCID identifiers.

u32_LCIDrange
Specifies whether GPI lcids or AVIO lcids or both are to be queried. *Note:* lcids -2, -3 and -4, represent AVIO lcids 1, 2 and 3 respectively (it is the function of the Engine to perform the remapping between these values and 1, 2 and 3 for the Device Driver).

1 GPI
2 AVIO
3 GPI & AVIO

Returns: *BOOL*

0 error
1 ok

SetBitmapID (u32_BmapH, u32_LCID, u32_DcH, u32_FuncNo)

This tags the specified bitmap with an LCID, so that the bitmap can be used for area shading or as the pattern in a BitBlt operation. When a bitmap is destroyed, its LCID becomes undefined.

If the specified LCID is already in use, then an error will be raised.

u32_BmapH
The handle of the bitmap.

u32_LCID
The LCID for which the definition is required.

Returns: *BOOL*

0 error
1 ok

SetCodePage (u32_CodePage, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_CodePage;
};
```

This sets the current code page.

u32_CodePage
: The new code page.

Returns: *BOOL*

0 error
1 ok

UnloadFont(p32_Filename, u32_FuncNo)

This unloads the font definitions which had been previously loaded from the specified resource file.

p32_Filename
: The filename of the font resource file.

Returns: *BOOL*

0 error
1 ok

GetCodepageTable (s32_cpid, p32_tab, u32_FuncNo)

The use of Multi-Codepage fonts implies that a translation is done from the codepoints of a character string (in one particular codepage) to the indices of the same character glyphs in the font. This is done via a Codepage Lookup table.

Codepage Lookup tables are available for the codepages 500, 850, 860, 863 and 865. They are provided in the Presentation Manager system by the Graphics Engine and are available to Device Drivers via this call.

Parameters:

s32_cpid
: is the codepage ID.

p32_tab
: is a pointer to a data area where the table is returned.

  The table is a simple list of 256 unsigned 16 bit values. Each value is the index number into the multi-codepage font of the glyph corresponding to the codepoint which addresses into the table (0-based).

u32_FuncNo
is the engine function number

Returns: *BOOL*

0 error
1 ok

GetRevCodeTable (s32_cpid, p32_tab, u32_FuncNo)

To ease conversion of text strings from one codepage to another, a further function is provided, which is essentially the reverse of GreGetCodepageTable.

Parameters:

s32_cpid
is the codepage ID.

p32_tab
is a pointer to a data area where the table is returned.

The table is a simple list of 300 unsigned 8 bit values. Each value is a codepoint value in the target codepage of the glyph corresponding to the index which addresses into the table (0-based).

u32_FuncNo
is the engine function number

Returns: *BOOL*

0 error
1 ok

16.1.12.18 Device Context Functions

The GPI_Prime is responsible for the creation/deletion of device contexts. Association of a Device Context with a Presentation Space (PS) and all Metafile functions are also provided by GPI_Prime.

Short list of the function calls:

- CopyDcLoadData
- OpenDevice Context
- CloseDevice Context
- GetDcOrigin
- PostDeviceModes
- GetHandle

- RestoreDC
- SaveDc
- SetDcOrigin
- SetHandle
- GetDCPointer

CopyDcLoadData(u32_Options, u32_SourceDcH, u32_DcH, u32_FuncNo)

Copies Line Type, Color Table and all lcid (i.e., Symbol Set, Logical Font, & Bitmap ID) related data from the source to the target DC.

Parameters:

u32_Options
Specifies whether to copy AVIO symbol sets only (LCIDs < X'41'), GPI symbol sets (LCIDS >= X'41') or both of these.

1 GPI
2 AVIO
3 GPI & AVIO

u32_SourceDcH
Specifies the handle of the source DC for the copy.

u32_DcH
Specifies the handle for the target DC for the copy.

Returns: *BOOL*

0 error
1 ok

OpenDC (u32_Type ,p32_Token, u32_Length, p32_Data, u32_DcH, u32_FuncNo)

Creates an output Device Context of a specified type.

The data passed depends upon the type of Device Context being created. It provides information such as the driver name, and may also provide data with which the Device Context is to be initialized.

Bits 0 & 1 of DC handle contains flags. which are ignored by the engine. They are set and used by the GPI layer as follows:

— *Bit 0* '1'B specifies Micro PS

— *Bit 1* '1'B specifies Metafile DC

Parameters:

u32_Type
: The type of Device Context to be created, as follows:

  2 - OD_QUEUED
  : A device such as a printer or plotter, for which output is to be queued by the spooler.

  5 - OD_DIRECT
  : A device such as a printer or plotter. Output is not queued by the spooler.

  6 - OD_INFO
  : As OD_DIRECT, but will only be used to retrieve information (for example, font metrics). Drawing can be performed to a presentation space associated with such a Device Context, but no output medium will be updated.

  8 - OD_MEMORY
  : A Device Context which will be used to contain a bitmap.

p32_Token
: A long pointer to a string which identifies device information, held in the *WIN.INI* file. This information is the same as that which may be pointed to by *p32_Data*; any that is obtained in this way overrides the information obtained by using *p32_ Token.*

  If the *token* is specified as "*" then no device information is taken from *WIN.INI. Note:* Presentation Manager Release 1 will ignore this parameter.

u32_Count
: The count of the number of long pointers present in *data.* This may be shorter This may be shorter than the full list if omitted items are irrelevant or supplied from *p32_ token* or elsewhere.

p32_Data
: A long pointer to a parameter block containing:

```
struct DOPDATA
     PSZ  address;
     PSZ  driver_name;
     PBYTE driver_data;
     PSZ  data_type;
     PSZ  comment;
     PSZ  proc_name;
     PSZ  proc_params;
     PSZ  spl_params;
     PSZ  network_params;
    ;
```

p32_address
: The address of the output device.

  - For a OD_DIRECT device, this is required if it is not available from the *token*. and consists of a logical device address, such as "LPT1".
  - For a OD_QUEUED device, this is optional, since the spooler will provide a default if necessary, and consists of a spooler queue name.

p32_DriverNname
: A long pointer to a string containing the name of the device driver. This information must always be supplied if it is not available from the *token*.

p32_DriverData
: A long pointer to data which is to be passed directly to the device driver. Whether or not any of this is required depends upon the device driver, though the information may alternatively have been specified via DeviceMode.

  The data consists of the following:

```
struct DRIVDATA
     LONG length;
     LONG version;
     SZ szDeviceName;
     ULONG abGeneralData;
    ;
```

  u32_DrvDataLength
  : The length of the whole *driver_data* structure.

  u32_Version
  : The version number of the data. Version numbers are defined by particular device drivers.

  DeviceName
  : A string in a 32-byte field, identifying the particular device (model number, etc.). Again, valid values are defined by device drivers.

  GeneralData
  : Data as defined by the device driver.

p32_DataType
A long pointer to a null terminated Data-Type character string.

- For a OD_QUEUED device, *DataType* defines the type of data which is to be queued, as follows:
  - "Q_STD" - standard format
  - "Q_ESC" - escape format
  - "Q_RAW" - raw format

  Note that a device driver may define other data types. If the *DataType* is not specified for a OD_QUEUED device, the default is supplied by the device driver.

In the above case, *DataType* information is defaulted if not specified.

For any other device *type*, *DataType* is ignored.

p32_Comment
A long pointer to a natural language description of the file. This may, for example, be displayed for a OD_QUEUED device by the spooler to the end user. It is optional for any de- vice.

p32_ProcName
A long pointer to the name of the queue processor. This is only relevant for a OD_QUEUED device, and will normally be defaulted.

p32_ProcParams
A long pointer to a parameter string for the queue processor. This is only relevant for a OD_QUEUED device, and is optional.

spl_params
A long pointer to a parameter string for the Spooler, which is optional. This is only relevant for a OD_QUEUED device, and is optional. This has the following options, which must be separated by one or more blanks:

- FORM=f

  Specifies a forms code 'f'. This must be a valid forms code for the printer.

  If not specified, then the data is printed

on the forms in use when this print job is ready to be printed.

- PRTY=n

  Specifies a priority in the range 0-99, with 99 being the highest. If not specified, then a priority of 50 is used.

network_params
: network parameters. This is only relevant for a OD_QUEUED device, and is optional. An application would leave it to the Network Program to specify this parameter, since it is for use in such an environment.

u32_DcH
: When u32_Type is OD_MEMORY, u32_DcH specifies the handle of a DC to create a compatible Device Context for. If the given DcH is 0, a Device Context compatible with the DISPLAY device is created.

Returns: *HDC*

0 Error
!=0 Device context handle

CloseDC (u32_DcH, u32_FuncNo)

This deletes the specified Device Context.

Returns: *BOOL*

0 error
1 ok

GetDcOrigin(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

Returns the DC origin of the device context.

Parameters:

p32_xy An XY pair to be used for the DC origin specified in screen coordinates.

Returns: *BOOL*

0 error
1 ok

PostDeviceModes(p32_DriverData, p32_DriverName, p32_DeviceName, p32_LogAddr, u32_FuncNo)

This function causes a device driver to post a dialog box that allows the user to set options for the device, for example resolution, font cartridges, etc.

The function can be called first with a NULL data pointer to find out how much storage is needed for the data area. Having allocated the storage, the application then calls the function a second time for the data to be filled in.

The returned data can then be passed on OpenDC as DriverData.

Parameters:

p32_DriverData
: A long pointer to a data area, which on return will contain device data as defined by the driver.

  If this pointer is passed as NULL, then the size in bytes which the data area should be is returned.

  The format of the data is as follows:

  u32_Length
  : The length of the whole DriverData structure in bytes.

  u32_Version
  : The version number of the data. Version numbers are defined by particular device drivers.

  DeviceName
  : A string 32-bytes long, identifying the particular device (model number, etc.). Again, valid values are defined by device driver.

  GeneralData
  : Data as defined by the device driver.

p32_DriverName
: A long pointer to a string containing the name of the device driver

p32_DeviceName
: A long pointer to a string identifying the particular device (model number, etc.). Valid names are defined by device drivers.

p32_LogAddr
: The logical address of the output device (e.g., "LPT1").

Returns: *short int*

p32_DriverData pointer was NULL:
-1 Error
0 No settable options
>0 Size in bytes required for data area

p32_DriverData pointer was not NULL:
-1 Error
0 No device modes
1 OK

GetHandle(u32_index, u32_DcH, u32_FuncNo)

This returns the handle (may be GPI_PS, AVIO_PS or metafile) for the specified index.

Parameters:

u32_index
Specifies the index value of the handle in the range 0..3 . There is actually a maximum of three but a slot is reserved for future expansion.

Returns: *long int*

-1 error
!= -1 returned handle

RestoreDc(s32_DcId, u32_DcH, u32_FuncNo)

Restore the contents of a previously saved DC.

Parameters:

s32_DcId
Specifies the ID of the DC state to be restored, or, a negative value will indicate the number of DCs to be popped before the required one.

on

1. If the ID specified is positive (>=0) and does not exist on the stack of saved DC's, then an error is returned and the DC is not modified.
2. If the ID specified is negative (<0) and there are insufficient entries on the stack then an error is returned and the DC is not modified
3. If the required DC is on the stack, then as a result of this function all entries up to the required one are removed from the stack as part of the RestoreDC process.
4. An ID value of 1 will cause the DC stack to be reset, all entries are removed from the stack.

5. An ID value of 0 will result in an error, and the DC is not modified.

Returns: *BOOL*

0 error
1 ok

SaveDc(u32_DcH, u32_FuncNo)

Save the specified DC's state on a stack and return an integer uniquely identifying it so that it can be easily restored at a future date.

The following are saved:

- Current attributes
- Current transforms and clip window
- Current position
- Reference to selected clip region
- Any loaded logical color table
- References to any loaded logical fonts
- References to any loaded symbol sets
- References to any loaded line type set
- References to the regions created on the associated Device Context

The following is not saved:

- The visible region

Note that any resources which are referenced in a saved DC (e.g., clip region, logical fonts, symbol sets, line type set) should not be deleted.

Note also that ID's of saved DC state's are unique only on a per DC basis, that is other DC's may have saved states with the same ID.

Parameters:

Returns: *long int*

0 error
!=0 id of saved dc

SetDcOrigin(p32_xy, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_xy;
};
```

Sets the DC origin of the specified device context. Note that the device origin is 0,0 when the device context is created.

Parameters:

p32_xy An XY pair to be used for the DC origin specified in screen coordinates.

Note: When the DC origin is set the Engine will align the clip regions at all saved levels of the DC (if one is set). The Rao region will not be recomputed until the Engine receives an explicit call to ComputeRegions or SelectVisRegion. (Note that the Rao region is the region reflecting the visible area on the screen).

Returns: *BOOL*

0 error
1 ok

SetHandle(u32_Handle, u32_index, u32_DcH, u32_FuncNo)

This associates the specified handle with the specified Device Context. A maximum of four handles may be associated with a single Device Context simultaneously. In practice this figure will probably never exceed three (i.e., a VIO Alphanumerics PS handle, a GPI Graphics PS handle and a metafile handle). Note that index zero will be used by the Engine as the GpiH parameter for the GpiVectorSymbol call from the Engine back to GPI-Prime. Dissociation is accomplished by specifying a null handle for u32_Handle.

Parameters:

u32_Handle
Specifies the handle.

u32_index
Specifies the index value of the PS in the range 0..3 .

Returns: *BOOL*

0 error
1 ok

GetDCPointer(u32_DcH, u32_FuncNo)

This function can be dynalinked directly by a device driver and allows it to convert a DC handle into a far pointer to the DC structure.

This function allows device drivers to be written in high level languages.

Returns: *Device Context far **

```
 0 error
!=0 far pointer to the Device Context
```

16.1.12.19 Transform and Clipping Functions

Short list of function calls:

- SetModelXform
- GetModelXform
- SetXFormRect
- SetViewingLimits
- GetViewingLimits
- SetWindowViewportXform
- GetWindowViewportXform
- SetGlobalViewingTransform
- GetGlobalViewingTransform
- SetPageUnits
- GetPageUnits
- SetGraphicsField
- GetGraphicsField
- SetPageWindow
- GetPageWindow
- SetPageViewport
- GetPageViewport
- Convert
- QueryViewportSize

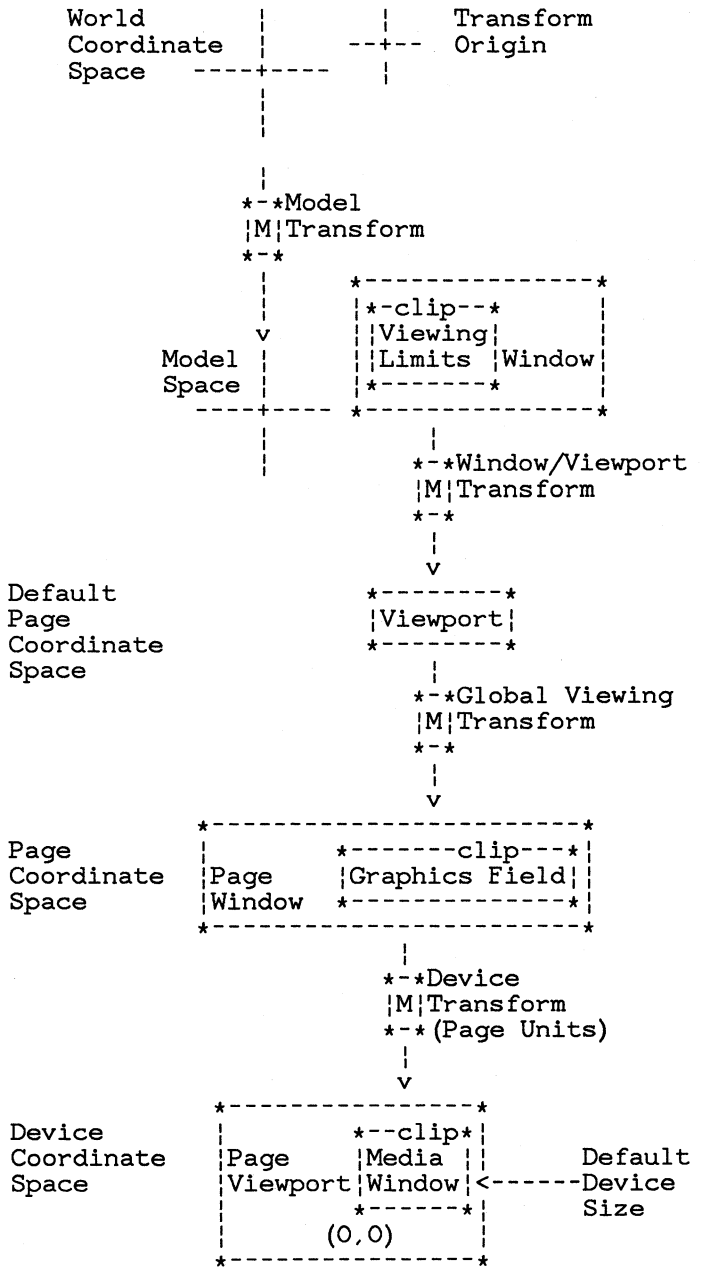


Figure 16.1 Engine Transformation and Clipping

SetModelXform(p32_XformData, u32_mode, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Mode;
        ULONG*  p32_XformData;
};
```

Sets the model transform matrix elements as specified.

Parameters:

p32_XformData
: Points to an array of 6 matrix elements for two-dimensional transformation:

  (fxM11, fxM12, fxM21, fxM22, M41, M42).

u32_mode
: Specifies how the supplied array should be used to set the matrix.

  Valid values are:

  0 Set unity transform (array values are ignored).
  1 Concatenate after (see u32_FuncNo for 'after' definition).
  2 Concatenate before (see u32_FuncNo for 'before' definition).
  3 Overwrite.

  Returns: *BOOL*

  0 error
  1 ok

GetModelXform(p32_XformData, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_XforData;
};
```

Returns an array of two-dimensional values which define the current model transform matrix.

Parameters:

p32_XformData
: Points to the return data area in which the array of 6 matrix elements is to be stored:

  (fxM11, fxM12, fxM21, fxM22, M41, M42),

Returns: *BOOL*

0 error
1 ok

SetXFormRect (p32_Rect, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Rect;
};
```

SetViewingLimits(p32_ViewLimits, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_ViewLimits;
};
```

Sets the boundaries of the viewing (clip) limits in model space to the specified values.

Parameters:

p32_ViewLimits
Points to a 4 element array containing the integer values that identify respectively the min_x, min_y, max_x and max_y boundaries of the viewing limits:

(s32_x1, s32_y1, s32_x2, s32_y2).

Returns: *BOOL*

0 error
1 ok

GetViewingLimits(p32_ViewLimits, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_ViewLimits;
};
```

Returns a 4 element array containing the integer values that identify the boundaries of the viewing window in graphic model space coordinates:

Parameters:

p32_ViewLimits
: Points to the return data area in which the array of 4 elements is to be stored. These are integer values that identify respectively the min_x, min_y, max_x and max_y boundaries of the viewing limits:

  (s32_x1, s32_y1, s32_x2, s32_y2).

  Returns: *BOOL*

  0 error
  1 ok

SetWindowViewportXform(p32_Transform, u32_mode, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Mode;
        ULONG*  p32_Transform;
};
```

Sets the Window/Viewport Transform matrix elements as specified.

Parameters:

p32_Transform
: Points to an array of 6 matrix elements for two-dimensional transformation.

  (fxM11, fxM12, fxM21, fxM22, M41, M42),

u32_mode
: Specifies how the supplied array should be used to set the matrix.

  Valid values are:

  0 Set unity transform (array values are ignored).
  1 Concatenate after (see u32_FuncNo for 'after' definition).
  2 Concatenate before (see u32_FuncNo for 'before' definition).
  3 Overwrite.

  Returns: *BOOL*

  0 error
  1 ok

GetWindowViewportXform(p32_Transform, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Transform;
};
```

Returns an array of two-dimensional values which define the current Window/Viewport transform matrix.

Parameters:

p32_Transform
: Points to the return data area in which the array of 6 elements is to be stored:

  (fxM11, fxM12, fxM21, fxM22, M41, M42),

  Returns: *BOOL*

  0 error
  1 ok

SetGlobalViewingXform(p32_Transform, u32_mode, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Mode;
        ULONG   p32_Transform;
};
```

Sets the Global Viewing Transform matrix elements to the specified values.

Parameters:

p32_Transform
: Points to an array of 6 matrix elements for two-dimensional transformation.

  (fxM11, fxM12, fxM21, fxM22, M41, M42),

u32_mode
: Specifies how the supplied array should be used to set the matrix.

  Valid values are:

  0 Set unity transform (array values are ignored).
  1 Concatenate after (see u32_FuncNo for 'after' definition).

2 Concatenate before (see u32_FuncNo for 'before' definition).
3 Overwrite.

Returns: *BOOL*

0 error
1 ok

GetGlobalViewingXform(p32_Transform, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   p32_Transform;
};
```

Returns an array of two-dimensional values which define the Global Viewing transform matrix.

Parameters:

p32_Transform
Points to the return data area in which the array of 6 matrix elements is to be stored:

(fxM11, fxM12, fxM21, fxM22, M41, M42),

Returns: *BOOL*

0 error
1 ok

SetPageUnits(u32_units,u32_width,u32_height,u32_DcH,u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Height;
        ULONG   u32_Width;
        ULONG   u32_Units;
};
```

This sets the page units controlling the Device Transform.

Parameters:

u32_units
Page Units, as follows:

Bits 0-1 Reserved, must be preserved by the Engine and returned by GetPageUnits

Bits 2-7

B'000001' Isotropic
Arbitrary units, as defined by u32_height and u32_width. 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 (maximized window size, paper size, etc.) and the origin at the bottom left.

B'000010' PelsUp
Pel coordinates, with the origin at the bottom left.

B'000011' LoMetric
Units of 0.1 mm, with the origin at the bottom left.

B'000100' HiMetric
Units of 0.01 mm, with the origin at the bottom left.

B'000101' LoEnglish
Units of 0.01 in, with the origin at the bottom left.

B'000110' HiEnglish
Units of 0.001 in, with the origin at the bottom left

B'000111' Twips
Units of 1/1440 in, with the origin at the bottom left.

Bits 8-31
Reserved, must be preserved by the Engine and returned by GetPageUnits

u32_width,u32_height
Specify the page width (w) and height (h).

A value of zero for w or h will cause it to be set to the corresponding default device dimension (maximized window size, paper size, etc.) in the specified page units (pels for isotropic).

This function causes the Window/Viewport Transform, Graphics Field, Page Window, Page Viewport and Device Transform to be updated (by the Engine) as follows:

For PelsUp, LoMetric, HiMetric, LoEnglish and HiEnglish:

Window/Viewport Transform Unity
Graphics Field (0,0) (w-1,h-1)
Page Window (0,0) (w-1,h-1)
Page Viewport (0,0) (sx*w-1,sy*h-1)
Device Transform As defined by Page Window, Page Viewport
Where sx = horizontal scaling required by page units for the device (= 1 for PelsUp)
Where sy = vertical scaling required by page units for the device (= 1 for PelsUp)

For Isotropic:

```
Window/Viewport Transform  Unity
Graphics Field             (0,0)  (w-1,h-1)
Page Window                  (0,0)  (w-1,h-1)
Page Viewport              (0,0)  (X2,Y2)
Device Transform           As defined by Page Window,
                                    Page Viewport
Where
   Dh is the default device (maximized window, etc.)
   height in pels.
   Dw is the default device (maximized window, etc.)
   width in pels.
   Wh is the page window height
     ( = ¦ (Y4 - Y3) ¦ + 1 where Y4 & Y3 are page
     window y coordinates)
   Ww is the page window width
     ( = ¦ (X4 - X3) ¦ + 1 where X4 & X3 are page
     window x coordinates)
   Par is the pixel (width/height) aspect ratio.
   X2, Y2 are integers determined as follows:
```

If Ww / Wh > Par * Dw / Dh then

```
X2  Dw-1
Y2  Par * Dw * Wh / Ww - 1
```

If Ww / Wh < Par * Dw / Dh then

```
X2  1/Par * Dh * Ww / Wh - 1
Y2  Dh-1
```

Otherwise (Ww / Wh = Par * Dw / Dh)

```
X2  Dw-1
Y2  Dh-1
```

Returns: *BOOL*

```
 0 error
 1 ok
```

GetPageUnits(p32_units, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Units;
};
```

This returns the page units for the specified display context. See SetPageUnits for a description of page units.

Parameters:

p32_units
: Points to the return data area in which the height and width are to be stored:

(u32_width, u32_height).

Returns: *long int*

0 error
>0 page units

SetGraphicsField(p32_GraphicsField, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_GraphicsField;
};
```

Sets the boundaries of the Graphics Field (clip) limits in Page coordinate space to the specified values.

Parameters:

p32_GraphicsField
: Points to a 4 element array containing the integer values that identify respectively the min_x, min_y, max_x and max_y boundaries of the graphics field:

(s32_x1, s32_y1, s32_x2, s32_y2).

Returns: *BOOL*

0 error
1 ok

GetGraphicsField(p32_GraphicsField, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_GraphicsField;
};
```

Returns a 4 element array containing the integer values that identify the boundaries of the graphics field.

Parameters:

p32_GraphicsField
: Points to the return data area in which the array of 4 elements is to be stored. These are integer values that identify respectively the min_x, min_y, max_x and max_y boundaries of the graphics field:

(s32_x1, s32_y1, s32_x2, s32_y2).

Returns: *BOOL*

0 error
1 ok

SetPageWindow(p32_Window, u32_Flags, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Flags;
        ULONG*  p32_Window;
};
```

This sets the Page Window in Page coordinate space for the Device Transform causing the Device Transform to be updated (by the Engine) using the Page Window and Page Viewport coordinates.

Parameters:

p32_Window
: Points to a 4 element array containing the integer values that identify respectively the boundaries of the window that correspond to the min_x, min_y, max_x and max_y viewport boundaries:

(s32_x1, s32_y1, s32_x2, s32_y2).

u32_Flags
: *Bit 0* Set to '1'B to indicate that the Device transform should be computed using the final Page Window and Page Viewport values. Set to '0'B to indicate that the Device Transform should not be modified.

Bit 1 Set to '1'B to indicate that the Page Viewport should be recomputed based on the Page Units (see SetPageUnits). Set to '0'B to indicate that the Page Viewport should not be modified.

Bits 2-31 Reserved.

Returns: *BOOL*

0 error
1 ok

GetPageWindow(p32_Window, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Window;
};
```

This returns the Page Window.

Parameters:

p32_window
: Points to the return data area in which the array of 4 elements is to be stored. These are integer values that identify respectively the boundaries of the window that correspond to the min_x, min_y, max_x and max_y viewport boundaries:

(s32_x1, s32_y1, s32_x2, s32_y2).

Returns: *BOOL*

0 error
1 ok

SetPageViewport(p32_Viewport, u32_Flags, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Flags;
        ULONG*  p32_Viewport;
};
```

This sets the Page Viewport in device coordinates causing the Device Transform to be updated (by the Engine) using the Page Window and Page Viewport coordinates.

Parameters:

p32_Viewport
: Points to a 4 element array containing the integer values that identify respectively the min_x, min_y, max_x and max_y boundaries of the page viewport:

(s32_x1, s32_y1, s32_x2, s32_y2).

u32_Flags

Bit 0 Set to '1'B to indicate that the Device transform should be computed using the final Page Window and Page Viewport values. Set to 'O'B to indicate that the Device Transform should not be modified.

Bits 1-31 Reserved.

Returns: *BOOL*

0 error
1 ok

GetPageViewport(p32_Viewport, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Viewport;
};
```

This returns the Page Viewport coordinates.

Parameters:

p32_Viewport
: Points to the return data area in which the array of 4 elements is to be stored. These are integer values that identify respectively the min_x, min_y, max_x and max_y boundaries of the page viewport:

(s32_x1, s32_y1, s32_x2, s32_y2).

Returns: *BOOL*

0 error
1 ok

Convert(u32_Source, u32_Target, p32_xy, u32_n, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_n;
        ULONG*  p32_xy;
        ULONG   u32_Target;
        ULONG   u32_Source;
};
```

Converts the specified coordinates from one coordinate space to another, using the current values of the transforms.

Parameters:

u32_Source, u32_Target
: Define the source and target coordinate spaces.

1 World coordinate space.
2 Model space.
3 Default Page coordinate space.
4 Page coordinate space.
5 Device coordinate space.
6 Screen coordinate space.

Screen coordinates have the format of 32 bit signed integers, and are used by the device driver as screen pixel addresses.

p32_xy Long pointer to an array of x,y coordinates to transform. The result is also put here.

u32_n Count of coordinate pairs in the array.

Returns: *BOOL*

0 error
1 ok

QueryViewportSize(u32_units, u32_count, p32_wh, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   p32_wh;
        ULONG   u32_Count;
        ULONG   u32_Units;
};
```

Note that this function is not required for the first release of Presentation Manager.

This calculates the viewport sizes (in page coordinate space) that, for the specified window sizes, produce a Window/Viewport transform that will emulate the specified u32_units with the current page units.

Parameters:

u32_units
As as defined for SetPageUnits

u32_count
The number of dimension pairs in p32_wh array

p32_wh
A far pointer to an array of u32_width, u32_height dimension pairs which are updated with the returned values.

Returns: *BOOL*

0 error
1 ok

16.1.12.20 Matrix Element Format

The format of the matrix elements for the model and viewing transforms are as follows:

fxM11, fxM12, fxM21 and fxM22

These are fixed point numbers with each element of the matrix represented as a signed 4-byte integer, with a notional binary point between bits 16 and 15. Thus

+2.5 is represented by*X'00028000'*
-2.5 is represented by*X'FFFD8000'*
-0.5 is represented by*X'FFFF8000'*

M41 and M42

These are signed 32 bit numbers.

16.1.12.21 Transform Definition by Window & Viewport

The matrix elements for a transform defined by Window and Viewport rectangles are determined as follows (where *X1, Y1, X2, Y2, X3, Y3, X4, Y4* represent the left, bottom, right and top coordinates of the viewport and the corresponding coordinates of the window respectively, X2 > X1 and Y2 > Y1 always):

```
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)
```

16.1.12.22 Bounds, Correlation and Clipping

Bounds computation is performed on unclipped primitives in model space. Bounds computation is performed on all operations that perform output to the device.

Correlation is performed on the output of primitives that have been clipped to the Viewing Limits and Graphics Field only in page coordinate space. Correlation is performed on all operations that perform output to the device and is inclusive of all boundaries of the pick window.

For clipping purposes, the transformed viewing limit rectangle can be approximated to the bounding rectangle of the minimum and maximum coordinates. Clipping is inclusive of all boundaries of the Viewing Limits and Graphics Field.

16.1.12.23 Logical Color Table Functions

Short list of function calls:

- CreateLogColorTable
- RealizeColorTable
- UnrealizeColorTable
- QueryColorData
- QueryLogColorTable
- QueryRealColors
- QueryNearestColor
- QueryColorIndex
- QueryRgbColor

CreateLogColorTable (u32_Options, u32_format, u32_Start, u32_Count, p32_Data, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Data;
        ULONG   u32_Count;
        ULONG   u32_Start;
        ULONG   u32_Format;
        ULONG   u32_Options;
};
```

This function defines the entries of the logical color table. The Engine will perform the error checking for CreateLogColorTable.

It may cause the color table to be preset to the default values. These are:

-2 White
-1 Black

0 Background (Black on display, White on printer)
1 Blue
2 Red
3 Pink (magenta)
4 Green
5 Turquoise (cyan)
6 Yellow
7 Neutral (White on display, Black on printer)

The range of color table indices (including the default color table) is -2..MaxIndex (not 0..MaxIndex).

Index -1 will never be loaded explicitly but will always produce the color value defined for for index 0 for a display or index 7 for a printer/plotter, etc. Index -2 will never be loaded explicitly but will always produce the color value defined for index 7 for a display or index 0 for a printer/plotter, etc.

Colors beyond 7 have device-dependent defaults.

Parameters:

u32_Options
: Specifies various options:

  LCOL_RESET (bit 0)
  : Set to B'1' if the color table is to be reset to default before processing the remainder of the data in this function

  LCOL_REALIZABLE (bit 1)
  : Set to B'1' if the application may issue RealizeColorTable at an appropriate time. This may affect the way the system maps the indices when the logical color table is not realized.

    If this option is not set, RealizeColorTable may have no effect

  LCOL_DITHER (bit 2)
  : Set to B'1' if the application does not want color dithering to be performed on colors not available in physical palette. If this option is set, only pure colors will be used and no dithering will be done.

    The default is color dithering.

  Other flags are reserved and must be B'0'.

u32_Format
: Specifies the format of entries in the table, as follows:

  LCOLF_INDRGB (1)
  : Array of (index,RGB) values. Each pair of

entries is 8 bytes long, 4 bytes (local format) index, and 4 bytes color value.

LCOLF_CONSECRGB (2)
Array of (RGB) values, corresponding to color indices *param* upwards. Each entry is 4 bytes long.

LCOLF_RGB (3)
Color index = RGB

u32_Start
Starting index (only relevant for LCOLF_CONSECRGB)

u32_Count
The number of elements supplied in *data*. This may be *0* if, for example, the color table is merely to be reset to the default, or for LCOLF_RGB. For LCOLF_INDRGB it must be an even number.

p32_Data
A pointer to the application data area, containing the color table definition data. The format depends on the value of *Format*.

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
```

(since there are 8 bits for each primary). The maximum intensity for each primary is 255.

The Engine will perform error checking for this function. Errors will include:

```
Insufficient Memory Available
Others - To Be Decided
```

Returns: *BOOL*

0 error
1 ok

RealizeColorTable (u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This function causes the system, if possible, to ensure that the device physical color table is set to the closest possible match to the logical color table.

Returns: *BOOL*

0 error
1 ok

UnrealizeColorTable (u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
};
```

This function is the reverse of RealizeColorTable. It causes the default color table to be reinstated.

Returns: *BOOL*

0 error
1 ok

QueryColorData (u32_Count, p32_Array, u32_DcH ,u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Array;
        ULONG   u32_Count;
};
```

Returns information about the currently available color table and device colors. *Note:* Default colors are not included in the loaded color index returned in Array[1] and Array[2].

Parameters:

u32_Count
The number of elements supplied in *Array.*

p32_Array
A pointer to, u32_Array, an array which on return will contain:

array[0] Format of loaded color table if any:

LCOLF_DEFAULT (0)
Default color table is in force.

LCOLF_INDRGB (1)
Color table loaded which provides translation from index to RGB.

LCOLF_RGB (3)
Color index = RGB.

Array[1]
Smallest color index loaded (*0* if the default color table is in force).

Array[2]
Largest color index loaded (*0* if the default color table is in force).

Array[3]
Maximum number of distinct colors available. at one time

Array[4]
Maximum number of distinct colors specifiable on device.

Array[5]
Maximum logical color table index supported for this device.

The range of logical color table indices is therefore -2..MaxIndex. The maximum index must be at least 7.

For EGA and u32_FuncNo, the value is 63.

Information will only be returned for the number of elements supplied. Any extra elements supplied will be zeroed by the system.

Returns: *BOOL*

0 error
1 ok

QueryLogColorTable (u32_Options, u32_Start, u32_Count, p32_array, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Array;
        ULONG"u32_Count;
        ULONG   u32_Start;
        ULONG   u32_Options;
};
```

Returns the logical color of the currently associated device, one at a time.

Parameters:

u32_Options
: Specifies various options:

  LCOLOPT_INDEX (bit 1)
  : Set to B'1' if the index is to be returned for each RGB value.

  Other flags are reserved and must be B'0'.

u32_Start
: The starting index for which data is to be returned.

u32_Count
: Specifies the number of elements available in the array pointed to by p32_array.

p32_array
: A pointer to an array in which the information is returned. If LCOLOPT_INDEX = B'0', this is an array of color values (each value is as defined for CreateLogColorTable), starting with the specified index, and ending either when there are no further loaded entries in the table, or when *u32_Count* has been exhausted. If the logical color table is not loaded with a contiguous set of indices, -1 will be returned as the color value for any index values which are not loaded.

  If LCOLOPT_INDEX = '1'B, it is an array of alternating color indices and values, in the order index1, value1, index2, value2,... If the logical color table is not loaded with a contiguous set of indices, any index values which are not loaded will be skipped.

Returns: *long int*

-4 error
-1 color table is in RGB mode and no elements are returned
>=0 number of elements returned

QueryRealColors (u32_Options, u32_Start, u32_Count, p32_Array, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_Array;
        ULONG"u32_Count;
        ULONG   u32_Start;
        ULONG   u32_Options;
};
```

Returns the RGB values of the distinct colors available on the currently associated device, one at a time.

Parameters:

u32_Options
: Specifies various options:

  LCOLOPT_REALIZED (bit 0)
  : Set to B'1' if the information required is to be for when the logical color table (if any) is realized; B'0' if it is to be for when it is not realized.

  LCOLOPT_INDEX (bit 1)
  : Set to B'1' if the index is to be returned for each RGB value.

  Other flags are reserved and must be B'0'.

u32_Start
: The ordinal number of the first color required. To start the sequence this would be 0. Note that this does not necessarily bear any relationship to the color index; the order in which the colors are returned is not defined.

u32_Count
: Specifies the number of elements available in the array pointed to by p32_array.

p32_Array
: A pointer to a u32_array in which the information is returned. If LCOLOPT_INDEX = B'0', this is an array of color values (each value is as defined for CreateLogColorTable). If LCOLOPT_INDEX = B'1', it is an array of alternating color indices and values, in the order index1, value1, index2, value2,... If there is a color table, colors that are not in the table but available on the device will have a special index of -5 when LCOLOPT_INDEX = B'1'. In RGB mode, when LCOLOPT_INDEX = b'1', the RGB value is returned in the color indices.

Returns: *long int*

```
-1 error
>=0 number of elements returned
```

QueryNearestColor (u32_Options, u32_RgbColorIn, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_RgbColorIn;
        ULONG   u32_Options;
};
```

Returns the nearest color available to the specified color, on the currently associated device, even if it is not available in the logical color table. Both colors are specified in RGB terms.

Parameters:

u32_Options
: Specifies various options:

  LCOLOPT_REALIZED (bit 0)
  : Set to B'1' if the information required is to be for when the logical color table (if any) is realized; B'0' if it is to be for when it is not realized.

  Other flags are reserved and must be B'0'.

u32_RgbColorIn
: The required color

Returns: *long int*

```
-1 error
>=0 nearest available RGB color to that specified
```

QueryColorIndex (u32_Options, u32_RgbColor, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_RgbColor;
        ULONG   u32_Options;
};
```

This returns the color index of the device color which is closest to the specified RGB color representation, for the specified device. In 'color index = RGB' mode, the input RGB value is returned.

Parameters:

u32_Options (ULONG)
Specifies various options:

LCOLOPT_REALIZED (bit 0)
Set to B'1' if the information required is to be for when the logical color table (if any) is realized; B'0' if it is to be for when it is not realized.

Other flags are reserved and must be B'0'.

u32_RgbColor
Specifies a color in RGB terms

Returns: *long int*

```
-1 error
>=0 color index providing closest match to specified color
```

QueryRGBColor (u32_Options, u32_Color, u32_DcH, u32_FuncNo)

```
place=inline frame = box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG   u32_Color;
        ULONG   u32_Options;
};
```

This returns the actual RGB color which will result from the specified color index, for the device specified. In 'color index = RGB' mode, the nearest RGB color (i.e., the same as QueryNearestColor) is returned.

Parameters:

u32_Options
Specifies various options:

LCOLOPT_REALIZED (bit 0)
Set to B'1' if the information required is to be for when the logical color table (if any) is realized; B'0' if it is to be for when it is not realized.

Other flags are reserved and must be B'0'.

u32_Color
Specifies a color index

Returns: *long int*

```
-1 error
>=0 nearest available RGB color to that specified
```

16.1.12.24 General Query Functions

Short list of the function calls:

- QueryEngineVersion
- QueryDeviceBitmaps
- QueryDeviceCaps
- QueryHardcopyCaps

QueryEngineVersion(u32_FuncNo)

This function returns the version number for the engine.

Parameters:

p32_Version
: A far pointer to u32_Version which returns the version number of the engine (this will return zero for the initial version of the engine).

Returns: *long int*

-1 error
>=0 Engine Vesion

QueryDeviceBitmaps(p32_OutData, u32_OutDataLength, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG  u32_OutDataLength;
        ULONG* p32_OutData;
};
```

This function returns a list of bitmap formats supported by the device. The number of formats supported can be found using the QueryDeviceCaps function. Each value in the list is of the form (u32_Planes, u32_BitsPerPixel).

The format at the start of the returned list is that which most closely matches the device.

Parameters:

p32_OutData
: A far pointer to the data structure to receive the data.

u32_OutDataLength
The length in bytes of the data structure pointed to by p32_OutData.

Returns: *BOOL*

0 error
1 ok

QueryDeviceCaps(u32_Index, p32_OutData, u32_Count, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_DcH;
        ULONG  u32_Count;
        ULONG* p32_OutData;
        ULONG  u32_Index;
};
```

This function returns information about the capabilities of the device.

Parameters:

u32_Index
Gives the index number of the first item of information to be returned in p32_OutData. The first element is number 1.

u32_Count
Gives the number of items of information to be returned at p32_OutData.

p32_OutData
A far pointer to an array of u32_Count elements (element type is s32) which, on return, will contain the elements specified by u32_Index and u32_Count. The first item returned is set into the first element of the array, the second into the next, and so on. The following index values are defined:

1 Device family (values as for *type* on OpenDC)

2 Device input/output capability

1 - Dummy device
2 - Device supports output
3 - Device supports input
4 - Device supports output and input

3 Technology

0 - Unknown (e.g., metafile)

1 - Vector plotter
2 - Raster display
3 - Raster printer
4 - Raster camera

4 Driver version

5 Default page depth (for a full-screen maximized window for displays) in display points. (For a plotter, a display point is defined as the smallest possible displacement of the pen, and can be smaller than a pen width.)

6 Default page width (for a full-screen maximized window for displays) in display points

7 Default page depth (for a full-screen maximized window for displays) in character rows

8 Default page width (for a full-screen maximized window for displays) in character columns

9 Vertical resolution of device in display points per meter for displays, plotter units per meter for plotters.

10 Horizontal resolution of device in display points per meter for displays, plotter units per meter for plotters.

11 Default character-box height in display points.

12 Default character box width in display points.

13 Default small character box height in display points (this is zero if there is only one character box size)

14 Default small character box width in display points (this is zero if there is only one character box size)

15 Number of distinct colors supported at the same time, including background (grayscales count as distinct colors). If loadable color tables are supported, this is the number of entries in the device color table.

For plotters, the returned value is the number of pens plus 1 (for the background).

16 Number of color planes

17 Number of adjacent color bits for each pel (within one plane)

18 Loadable color table support:

Bit 0 - 1 if RGB color table can be loaded, with a minimum support of 8 bits each for red green and blue
Bit 1 - 1 if color table with other than 8 bits for each primary can be loaded

19 The number of mouse or tablet buttons that are available to the application program. A returned value of 0 indicates that there are no mouse or tablet buttons available.

20 Foreground mix support

1 - OR
2 - Overpaint
4 - Underpaint
8 - Exclusive-OR
16 - Leave alone
32 - AND
64 - Mixes 7 thru 17

The value returned is the sum of the values appropriate to the mixes supported. A device capable of supporting OR must, as a minimum, return 1 + 2 + 16 = 19, signifying support for the mandatory mixes OR, overpaint, and leave-alone.

Note that these numbers correspond to the decimal representation of a bit string that is seven bits long, with each bit set to 1 if the appropriate mode is supported.

21 Background mix support

1 - OR
2 - Overpaint
4 - Underpaint
8 - Exclusive-OR
16 - Leave alone

The value returned is the sum of the values appropriate to the mixes supported. A device OR must, as a minimum, return 2 + 16 = 18, signifying support for the mandatory background mixes overpaint, and leave-alone.

Note that these numbers correspond to the decimal representation of a bit string that is five bits long, with each bit set to 1 if the appropriate mode is supported.

22 Number of symbol sets which may be loaded for alphanumerics.

23 Whether the client area of Vio windows should be byte-aligned:

0 - Must be byte-aligned
1 - More efficient if byte-aligned, but not required
2 - Does not matter whether byte-aligned

24 Number of bitmap formats supported by device

25 Device raster operations capability

Bit 0 - 1 if BitBlt supported
Bit 1 - 1 if this device supports banding
Bit 2 - 1 if BitBlt with scaling supported
Bit 3 - 1 if FloodFill supported
Bit 4 - 1 if SetPel supported

26 Default marker box width in pels

27 Default marker box depth in pels

28 Number of device specific fonts

29 Graphics drawing subset supported

30 Graphics architecture version number supported (1 indicates Version 1)

31 Graphics vector drawing subset supported

32 Device windowing support

Bit 0 - 1 if Device supports windowing

Other bits are reserved zero.

33 Additional graphics support

Bit 0 - 1 if Device supports geometric line types

Other bits are reserved zero.

Returns: *BOOL*

0 error
1 ok

QueryHardcopyCaps(u32_Start, u32_Count, p32_Info, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG  u32_FuncNo;
        ULONG  u32_Dch;
        ULONG* p32_Info;
        ULONG  u32_Count;
        ULONG  u32_Start;
};
```

This function returns information about the hardcopy capabilities of the device.

Parameters:

u32_Start
: Specifies which forms code number the query is to start from, where the first forms code is specified by the value 1. Used with *count.*

u32_Count
: Specifies the number of forms the query is to be on. Thus if there are 5 form codes defined and *start* is 2, then if *count* is 3, a query is performed for form codes 2, 3 and 4, and the result returned in the buffer pointed to by *p32_Info.*

  If this value is zero, the number of form codes defined is returned. If non-zero (i.e., greater than zero), the number of form codes information was returned for is returned.

p32_Info
: Pointer to a buffer containing the results of the query. The result consists of *count* copies of the following structure:

```
struct HCINFO
     CHAR szFormname[32];
     LONG cx;
     LONG cy;
     LONG xLeftClip;
     LONG yBottomClip;
     LONG xRightClip;
     LONG yTopClip;
     LONG xPels;
     LONG yPels;
     LONG attributes;
    ;
```

szFormname
: The ASCIIZ name of the form.

cx
: The width (left to right) in millimeters.

cy
: The height (top to bottom) in millimeters.

xLeftClip
: The left clip limit in millimeters.

yBottomClip
: The bottom clip limit in millimeters.

xRightClip
: The right clip limit in millimeters.

yTopClip
: The 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.

attributes
: Attributes of the form, defined as follows:

  Bit 0 `- 1 if current installed form`

Note: start and *count* can be used together to enumerate all available form codes without having to allocate a buffer large enough to hold information on them all.

Returns: *long*

```
-1 Error
>=0 If count == 0, number of forms available
>=0 If count != 0, number of forms returned
```

16.1.12.25 Escape Functions

Short list of the function calls:

- Escape

Escape(u32_Escape, u32_InCount, p32_InData, p32_OutCount, p32_OutData, u32_DcH, u32_FuncNo)

```
place=inline frame=box

struct ARGUMENTS {
        ULONG   u32_FuncNo;
        ULONG   u32_DcH;
        ULONG*  p32_OutData;
        ULONG*  p32_OutCount;
        ULONG*  p32_InData;
        ULONG   u32_InCount;
        ULONG   u32_Escape;
};
```

This function allows applications to access facilities of a particular device that are not directly available through the GPI. Escape calls are in general sent to the device driver and must be understood by it.

Parameters:

u32_Escape
: Specifies the escape function to be performed. The following predefined functions are available.

  - 1 - QueryEscSupport
  - 2 - StartDoc
  - 3 - EndDoc
  - 4 - NewFrame
  - 5 - NextBand
  - 6 - AbortDoc
  - 7 - DraftMode
  - 8 - GetScalingFactor
  - 9 - FlushOutput
  - 10 - RawData

  Devices can define additional escape functions, using *code* values > 32767.

u32_InCount
: Specifies the number of bytes of data pointed to by p32_InData.

p32_InData
: A far pointer to the input data structure for this escape.

p32_OutCount
: A far pointer to u32_count which on input specifies the size of the buffer pointed to by p32_Outdata and on output is set to the number of data bytes returned in this buffer.

p32_OutData
: A far pointer to the data structure to receive data from this escape.

Returns: *long*

```
-1 Error
 0 Escape not implemented for specified code
 1 OK
```

The predefined escape functions are described below:

QueryEscSupport

This function finds out whether a particular escape function is implemented by the device driver. The return value gives the result.

Parameters:

u32_InCount
: The number of bytes pointed to by *in_ data.*

p32_InData
: Pointer to an escape code value specifying the escape function to be checked.

p32_OutCount
: Not used, and can be set to zero.

p32_OutData
: Not used, and can be set to null.

StartDoc

This function allows an application to specify that a new print job is starting and that all subsequent *NewFrame* calls should be spooled under the same job, until an *EndDoc* call occurs.

This ensures that documents longer than one page are not interspersed with other jobs.

Parameters:

u32_InCount
: Specifies the number of characters in the string pointed to by *p32_ InData.*

p32_InData
: Pointer to an ASCIIZ string, specifying the name of the document.

p32_OutCount
: Not used, and can be set to zero.

p32_OutData
: Not used, and can be set to null.

EndDoc

This function ends a print job started by *StartDoc* and returns the jobid for the spooled print job.

Parameters:

u32_InCount
: Not used, and can be set to zero.

p32_InData
: Not used, and can be set to null.

p32_OutCount
: A far pointer to u32_count which on input specifies the size of the buffer pointed to by p32_Outdata and on output is set to the number of data bytes returned in this buffer (i.e., zero if no jobid).

p32_OutData
: A long pointer to a data area in which the jobid of the spooled print job is returned. This is set to null if there is no jobid (e.g., for a direct printer).

NewFrame

This function allows an application to specify that it has finished writing to a page. It is similar to *ErasePS* processing for a Screen DC, and causes a reset of the attributes (e.g., color, mix). This escape is usually used with a printer device to advance to a new page.

Parameters:

u32_InCount
: Not used, and can be set to zero.

p32_InData
: Not used, and can be set to null.

p32_OutCount
: Not used, and can be set to zero.

p32_OutData
: Not used, and can be set to null.

NextBand

This function allows an application to specify that it has finished writing to a band. The coordinates of the next band are returned. This is used by applications that handle banding themselves.

Parameters:

u32_InCount
Not used, and can be set to zero.

p32_InData
Not used, and can be set to null.

p32_OutCount
Specifies the number of bytes of data pointed to by *p32_OutData*. On return, this is updated to the number of bytes actually returned.

p32_OutData
The address of a buffer which will receive the output from this escape. A structure is returned, containing the device coordinates of the next band, which is a rectangle. The format of the structure is:

```
struct BANDRECT
     LONG xleft;
     LONG ytop;
     LONG xright;
     LONG ybottom;
    ;
```

xleft — The x coordinate of the upper left corner of the rectangular band.

ytop — The y coordinate of the upper left corner of the rectangular band.

xright — The x coordinate of the lower right corner of the rectangular band.

ybottom
The y coordinate of the lower right corner of the rectangular band.

An empty rectangle (i.e., xleft > xright, ytop < ybottom) marks the end of the banding operation.

AbortDoc

This function aborts the current job, erasing everything the application has written to the device since the last *EndDoc*.

Parameters:

u32_InCount
Not used, and can be set to zero.

p32_InData
Not used, and can be set to null.

p32_OutCount
Not used, and can be set to zero.

p32_OutData
Not used, and can be set to null.

DraftMode

This function turns draft mode on or off. Turning it on instructs the device driver to print faster and with lower quality, if necessary. The draft mode can only be changed at page boundaries (e.g., after a *NewFrame*).

Parameters:

u32_InCount
Specifies the number of bytes pointed to by *p32_InData*.

p32_InData
A long pointer to a SHORT integer value specifying the mode: 1 for draft mode on, 0 for off.

p32_OutCount
Not used, and can be set to zero.

p32_OutData
Not used, and can be set to null.

GetScalingFactor

This function retrieves the scaling factors for the x and y axes of a printing device. For each scaling factor, an exponent of two is put in *P32_OutData*. Thus, the value 3 is used if the scaling factor is 8.

Scaling factors are used by devices that cannot support graphics at the same resolution as the device resolution.

Parameters:

u32_InCount
Not used, and can be set to zero.

p32_InData
Not used, and can be set to null.

p32_OutCount
Specifies the number of bytes of data pointed to by *P32_OutData*. On return, this is updated to the number of bytes actually returned.

p32_OutData
The address of a buffer which will receive the output from this escape. A structure is returned, containing the scaling factors for the x and y axes. The format of the structure is:

```
struct SFACTORS
     LONG x;
     LONG y;
    ;
```

x The x scaling factor, as an exponent of two.

y The y scaling factor, as an exponent of two.

FlushOutput

This function flushes any output in the device's buffer.

Parameters:

u32_InCount
Not used, and can be set to zero.

p32_InData
Not used, and can be set to null.

p32_OutCount
Not used, and can be set to zero.

p32_OutData
Not used, and can be set to null.

RawData

This function allows an application to send data direct to a device driver. For example, in the case of a printer device driver, this could be a printer data stream.

Parameters:

u32_InCount
The number of bytes pointed to by *p32_InData.*

p32_InData
Pointer to the raw data. to be checked.

p32_OutCount
Not used, and can be set to zero.

p32_OutData
Not used, and can be set to null.

16.1.12.26 Enable Function

The Enable function is required by the Device Driver Interface.

The Enable function is exported by the device driver. It performs initialization of the device driver, the physical device, and device contexts. It will be called as:

Enable(U32_SUBFUNCTION , P32_PARAMS , P32_RETURNS)

16.1.12.26.1 U32_SUBFUNCTION = 1 Fill lDeviceBlock

Initializes the logical device block. This function will be called whenever the device driver module is loaded.

Parameters:

P32_PARAMS
: Pointer to a structure as follows:

  U32_VERSION
  : Version of the Graphics Engine. This is a BCD coded version number.

  U32_TABLE_SIZE
  : The number of entries in the dispatch table. The device driver should not replace pointers past the end of the table as indicated by this number.

P32_RETURNS
: Pointer to a structure as follows:

  P32_FLAGS
  : Pointer to a word of logical device flags. The device driver should set bits 0, 1, and 2 of these flags. All other flags are reserved for system use and must not be modified. The bits are defined as follows:

    BIT 0 — Set if each DC for this device will require its own pDeviceBlock. Clear if only one pDeviceBlock is needed for each physical device. It is expected that printer and plotter drivers would set this bit, and most others would clear it.

    BIT 1 — Set if this device can have only one DC open at any time; This is a serially reusable device. Clear if an arbitrary number of DCs may coexist.

BIT 2 Set if the "device" and "file name" fields of a CreateDC call for this device should be ignored. This would be the case if the device driver supported only one physical device in one configuration, like the display driver, for example.

P32_DISPATCH_TABLE
Pointer to the dispatch table. Each entry in the table is a 32 bit pointer to a major function handler. This table is already filled with the addresses of the system default handlers when this call is made. The device driver must replace the entries in the table that correspond to required major function handlers (see relevant section). The device driver may replace more entries, at its option. This table will then be used to dispatch major function handlers for ALL physical devices belonging to this logical device.

16.1.12.26.2 U32_SUBFUNCTION = 2 Fill pDeviceBlock

Initializes a physical device block. This may be called once per physical device or once per DC allocation, depending on how the device driver responded with BIT 0 of the ldb_flags on the lDeviceBlock call.

Parameters:

P32_PARAMS
Pointer to a structure defined as:

p32_drivername
Pointer to ASCIIZ name of the driver.

p32_devicename
Pointer to ASCIIZ name of the device.

p32_outputname
Pointer to ASCIIZ name of the output device. This may be either the spooler output class (e.g., "PRINT" or "PLOT"), or the device name for the physical port (e.g., "LPT1").

p32_devicedata
Pointer to device specific initialization data.

u32_datatype
One of:

1 => Device Independant
2 => Device Dependent
3 => Raw

4 => Default
5 => Device Driver

P32_RETURNS
Pointer to pDeviceBlock structure.

U16_LENGTH
Length in bytes of the pDeviceBlock structure. The device driver must not change this field.

U16_FLAGS
Physical device flags. All flags are reserved for system use and must not be modified.

U32_COUNT
Reference count for this pDeviceBlock. The device driver must not change this field.

U32_NEXT
Pointer to the next pDeviceBlock belonging to the same logical device. The device driver must not change this field.

U32_DEVICE
Atom for the name of the physical device, like "FX-80". The device driver must not change this field.

U32_FILE
Atom for the file name of the port this device is connected to, like "LPT1". The device driver must not change this field.

U32_STATEINFO
Pointer or handle for the state information for this device. The device driver should allocate its own memory for this purpose and use this field later to locate it.

16.1.12.26.3 U32_SUBFUNCTION = 3 Fill Information pDeviceBlock

Fills a pDeviceBlock that will never be used to perform actual drawing. It is used for information retrieval only.

Parameters:

P32_PARAMS
Pointer to a structure defined as:

p32_drivername
Pointer to ASCIIZ name of the driver.

p32_devicename
Pointer to ASCIIZ name of the device.

p32_outputname
: Pointer to ASCIIZ name of the output device. This may be either the spooler output class (e.g., "PRINT" or "PLOT"), or the device name for the physical port (e.g., "LPT1").

p32_devicedata
: Pointer to device specific initialization data.

u32_datatype
: One of:

  1 => Device Independant
  2 => Device Dependent
  3 => Raw
  4 => Default
  5 => Device Driver

P32_RETURNS
: Pointer to pDeviceBlock structure.

U16_LENGTH
: Length in bytes of the pDeviceBlock structure. The device driver must not change this field.

U16_FLAGS
: Physical device flags. All flags are reserved for system use and must not be modified.

U32_COUNT
: Reference count for this pDeviceBlock. The device driver must not change this field.

U32_NEXT
: Pointer to the next pDeviceBlock belonging to the same logical device. The device driver must not change this field.

U32_DEVICE
: Atom for the name of the physical device, like "FX-80". The device driver must not change this field.

U32_FILE
: Atom for the file name of the port this device is connected to, like "LPT1". The device driver must not change this field.

U32_STATEINFO
: Pointer or handle for the state information for this device. The device driver should allocate its own memory for this purpose and use this field later to locate it.

16.1.12.26.4 U32_SUBFUNCTION = 4 Disable pDeviceBlock

Any physical disabling of the specified device is performed and any associated memory is deallocated.

Parameters:

P32_PARAMS
: Ignored for this subfunction.

P32_RETURNS
: Pointer to pDeviceBlock structure.

  U16_LENGTH
  : Length in bytes of the pDeviceBlock structure.

  U16_FLAGS
  : Physical device flags.

  U32_COUNT
  : Reference count for this pDeviceBlock.

  U32_NEXT
  : Pointer to the next pDeviceBlock belonging to the same logical device.

  U32_DEVICE
  : Atom for the name of the physical device, like "FX-80".

  U32_FILE
  : Atom for the file name of the port this device is connected to, like "LPT1".

  U32_STATEINFO
  : Pointer or handle for the state information for this device.

16.1.12.26.5 U32_SUBFUNCTION = 5 Enable Device Context

This function will be called when a new DC is created. The device driver is expected to allocate any memory it needs to support the attributes of the DC. It then should store a handle for this memory in the DC "magic number".

Parameters:

P32_PARAMS
: Pointer to pDeviceBlock structure.

P32_RETURNS
: Pointer to the new Device Context. The only information the device driver has about the DC structure is that the magic number is at offset 4. That is, to the device driver, the DC structure is as follows:

U32_RESERVED
Reserved. The device driver must not modify this field.

U32_MAGIC
This field is under the complete control of the device driver. When this subfunction is called the field is initialized to zero. The device driver is expected to store here enough information to locate its instantiations of any of this DC's attributes.

U32_RESERVED[many]
Reserved. The device driver must not modify this field.

16.1.12.26.6 U32_SUBFUNCTION = 6 Disable Device Context

This function will be called when a DC is about to be deleted. The device driver is expected to free up any memory it has allocated for the DC. It is expected that the device driver will use the "magic number" in the DC to locate this memory.

Parameters:

P32_PARAMS
Pointer to the DC structure.

P32_RETURNS
Ignored for this subfunction.

16.1.12.26.7 U32_SUBFUNCTION = 7 Save DC State

This function will save a copy of whatever information the device driver has stored about this DC. A DC's state may be saved multiple times, in a LIFO order. This function will return an error code if there is not enough memory available to save the state.

Parameters:

P32_PARAMS
Pointer to the Device Context whose state is to be saved.

P32_RETURNS
Pointer to a 32 bit count. As a return value, the count will be set to the number of states that are saved for this DC. Later on, this number can be used with the RESTORE DC STATE call to restore the state we have just saved. If P32_RETURNS is NULL, then no count will be returned.

16.1.12.26.8 U32_SUBFUNCTION = 8 Restore DC State

This function will restore a previously saved DC state. A parameter to this function is the number of saved states that should be "POPed". This function will return an error code if is has been asked to POP more states than have been PUSHed.

Parameters:

P32_PARAMS
: This is a number indicating what state should be restored. If the number is positive, it indicates which state in the order they were PUSHed. That is, if the number is one, then the first PUSHed state is restored, and all others are lost. If the number is two, The second PUSHed state will be restored, and one will remain saved. If the number is negative, it indicates how many states will be POPed. That is, if the number is -1, we will POP back one state. If the number is zero, an error will be returned. If a positive or negative number is given specifying a state that hasn't been saved, an error will be returned.

P32_RETURNS
: Pointer to the Device Context whose state is to be restored.

16.1.12.26.9 U32_SUBFUNCTION = 9 Reset DC State

This function will reset the information saved for this DC to its original initialized state.

Parameters:

P32_PARAMS
: Ignored for this subfunction.

P32_RETURNS
: Pointer to the Device Context whose state is to be reset.

16.1.12.26.10 U32_SUBFUNCTION = 10 Disable display output

This function will be called only for a display driver. The call will be made, for example, when the screen group is switched. The device driver should not do any writing to the physical display after receiving this call, until the ENABLE DISPLAY OUTPUT call is made. The device driver may want to save any state of the display hardware that may be destroyed by another screen group.

Parameters:

P32_PARAMS
Ignored for this subfunction.

P32_RETURNS
Ignored for this subfunction.

16.1.12.26.11 U32_SUBFUNCTION = 11 Enable display output

This function will be called only for a display driver. The call will be made, for example, when the Presentation Manager screen group is restored. The device driver should restore the state of the display device. It may then resume output to the display.

Parameters:

P32_PARAMS
Ignored for this subfunction.

P32_RETURNS
Ignored for this subfunction.

16.1.12.26.12 U32_SUBFUNCTION = 12 Install Simulation

This function will be called only for an installable simulation. This is the only subfunction that an installable simulation needs to handle.

The simulation is expected to do any initialization that it needs. It must also place pointers to its own major functions in the given dispatch table. It may wish to record the pointers that it is overwriting in case it does not completely handle the major function.

The simulation should return zero if the installation was successful. Otherwise, it should return ERROR_WRONG_VERSION or ERROR_COMPONENT_NOT_FOUND.

Parameters:

P32_PARAMS
A pointer to the following structure:

U32_VERSION
The BCD coded engine version number.

P32_COMPONENT
A pointer to the ASCIIZ string indicating which component to install. By using these component names, a single file on the disk can contain the code for several simulations, like: "REGIONS", "ARCS", or

"TRANSFORMS". Even if a file contains only one simulation component, it should check the name for consistency.

U32_TABLE_SIZE
: The number of entries in the dispatch table. The simulation should not replace pointers past the end of the table as indicated by this number.

P32_RETURNS
: A pointer to the major function dispatch table. Each entry in the table is a 32 bit pointer to a major function handler. The simulation should replace the entries in this table that it wants to handle. It may wish to record the previous handle's address in case it can't handle the function completely.

16.1.13 Journaling Functions

Short list of function calls:

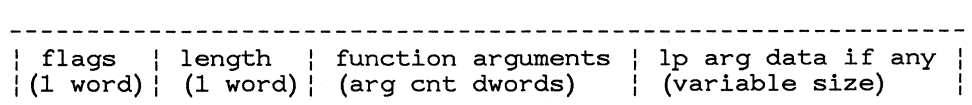
- AccumulateJournalFile
- CreateJournalFile
- DeleteJournalFile
- StartJournalFile
- StopJournalFile
- PlayJournalFile
- OpenJournalFile

Format of Engine Journal Files

The format of a journal record for a single function is:

```
flags          USHORT
length         USHORT      length in bytes of journaled record
function_args  ULONG * N   (N = number of args for function)
lparg_data     the data pointed to by any lpData type arguments
```

or, graphically:



Any lpData type arguments are converted to offsets to the journaled data before the record is written to disk and then fixed up at playback time.

If the journal file is a permanent journal file and the function was Bitblt (with a Source DC), SetBitmapID, PaintRegion or SelectClipRegion there is additional data following the journal record in the form:

```
length          ULONG
rects_or_bits   USHORT  as many words as necessary
```

The rects or bits field for bitmaps includes a bitmap info and color table.

AccumulateJournalFile (?, hDC, FunN)

The arguments of this function will vary according to what Engine function is to be journaled.

This function constructs and writes to disk a journal record for the function being journaled. The format of the record is:

```
-------------------------------------------------------------
| flags   | length   | function arguments | lp arg data if any |
|(1 word) | (1 word) | (arg cnt dwords)   | (variable size)    |
-------------------------------------------------------------
    LOW memory
```

The lp arg data is the data pointed to by any lp arguments in the function argument list. The actual journaled lp argument is changed into an offset to the journal lp arg data and is fixed up at playback time.

Returns: return value of function accumulated

CreateJournalFile (lpFileName, FunN)

This functions creates a disk journal file. If lpFileName is NULL (0:0) a temporary file is created in the current directory. If the pointer is not null it is assumed to point to a fully qualified file name and this file is created and opened. Such a file is referred to below as a permanent journal file.

Returns: *HJOURNAL*

DeleteJournalFile (hJournalFile, FunN)

This function frees any objects associated with the journal file handle (compatible DCs, private clone regions or bitmaps, global memory segments, etc.). If the handle refers to a temporary file the file is also deleted. Finally, the file handle itself is freed.

Returns: *BOOL*

StartJournalFile (hDC, hJournalFile, FunN)

Starts the journaling process. Opens the previously created journal file and turns on the COM_RECORDING bit in the DC's dc_mode. Subsequent GreEntry calls to this DC will pass through AccumulateJournalFile until StopJournalFile is called.

Returns: *BOOL*

StopJournalFile (hDC, hJournalFile, FunN)

Writes the END_OF_JOURNALFILE marker into the journal file, closes the journal file and turns off the COM_RECORDING bit in the DC's dc_mode.

Returns: *BOOL*

PlayJournalFile (Control, hDC, hJournalFile, FunN)

This functions plays the specified journal file to the specified DC. The journal file is read into memory and and each journaled GreEntry call is played.

Each journaled record is processed before playing to the extent that long pointers to data are fixed up and clone objects (regions or bitmaps) are created if necessary from the journaled data.

It is assumed that any single journaled function and associated lp data fits in a 32K buffer. If the journaled record contains region rectangles or bitmap bits these do not count in this restriction.

Returns: *BOOL*

OpenJournalFile (lpFileName, hDC, FunN)

This functions opens for play a journal file previously created with CreateJournalFile and accumulated to with AccumulateJournalFile. This function is for use with permanent journal files. See the description of CreateJournalFile.

Returns: *HJOURNAL*

16.1.14 Area Support Functions

The Area filling component requires the following support functions. These belong to the Engine API but only as an internal interface. These functions are expected to be called only by the Area Simulation component. The function LineToShortLine will be handled by the Device Driver as a required function. The functions CurveToShortLine and IntersectCurves will be handled by the Arc Simulation component.

Short list of the functions

- CurveToShortLine
- LineToShortLine
- IntersectCurves

CurveToShortLine (p32_Curve, p32_Buffer, s32_Y1, s32_Y2, u32_DcH, u32_FuncNo)

Converts a curve to a PolyShortLine in the space provided at p32_Buffer. The PolyShortLine runs only between the given s32_Y1 and s32_Y2 coordinates. The format of the PolyShortLine is described in the writeup of the PolyShortLine call. The curve that must be rendered is indicated by p32_Curve which points to the following sharp fillet structure.

```
fillet_sharp_rec struc
    c_ident     USHORT      ; identifier for firewalls
    c_next      USHORT      ; pointer to next curve
    c_prev      USHORT      ; pointer to previous curve
    c_type      BYTE        ; type of this curve
    c_bits      BYTE        ; flags and scratch bits
    c_edge      USHORT      ; ptr to assoc contour edge
    c_a         ULONG       ; start point of conic
    c_c         ULONG       ; end point of conic
    c_b         ULONG       ; join point of conic
    c_sharp     ULONG       ; sharpness of conic
    c_eqn       USHORT      ; near pointer to fillet_eqn
                USHORT      ; unused
fillet_sharp_rec ends
```

The c_eqn field points to the actual equation for the conic:

```
fillet_eqn struc
    fe_ident        USHORT      ; identifier for firewalls
                    USHORT      ; unused
    fe_type         BYTE        ; type of this block
    fe_bits         BYTE        ; flags and scratch bits
    fe_alpha        BYTE        ; coefficient of x**2
    fe_beta         BYTE        ; coefficient of 2xy
    fe_gamma        BYTE        ; coefficient of y**2
    fe_delta        BYTE        ; coefficient of 2x
    fe_epsilon      ULONG       ; coefficient of 2y
    fe_zeta¦        BYTE        ; coefficient of 1
                    USHORT * 5  ; unused
fillet_eqn ends
```

Returns: *BOOL int*

0 error
1 ok

LineToShortLine (p32_Line, p32_Buffer, s32_Y1, s32_Y2, u32_DcH, u32_FuncNo)

Converts a line to a PolyShortLine in the space provided at p32_Buffer. The PolyShortLine runs only between the given s32_Y1 and s32_Y2 coordinates. The format of the PolyShortLine is described in the writeup of the PolyShortLine call. The line that must be rendered is indicated by p32_Curve which points to the following line description.

```
line_rec struc
    l_ident     USHORT      ; identifier for firewalls
    l_next      USHORT      ; pointer to next curve
    l_prev      USHORT      ; pointer to previous curve
    l_type      BYTE        ; type of this curve
    l_bits      BYTE        ; flags and scratch bits
    l_edge      USHORT      ; ptr to assoc contour edge
    l_a         ULONG       ; start point of line
    l_c         ULONG       ; end point of line
    l_rise      ULONG       ; delta y of line
    l_run       ULONG       ; delta x of line
    l_intcept   ULONG       ; x intercept of line
    l_rslope    ULONG       ; dx/dy
line_rec     ends
```

Returns: *BOOL int*

0 error
1 ok

IntersectCurves (p32_Curve1, p32_Curve2, p32_Points, u32_DcH, u32_FuncNo)

Will return up to two intersection points for the curves. The given curves, p32_Curve1 and p32_Curve2, are any combination of lines and conics, as described above. The intersection points are stored in the given area in fixed point notation.

Returns: *int*

-1 error
>=0 number of intersection points

16.1.15 Callback Functions

This section defines Engine calls that are available only at the DDI (note that GetCodepageTable and GetRevCodeTable are also callback functions available to device drivers).

Short list of the function calls:

- AccumulateBounds
- AllocateCurves
- ChangePolygonMode
- ClipLine
- ClipConic
- ClipRect
- ClipScans
- ClipPoly

- ClipPathCurves
- NotifyTransformChange

AccumulateBounds (p32_Rect, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
    ULONG   u32_FuncNo;
    ULONG   u32_DcH;
    ULONG*  p32_Rect
};
```

This function is used to pass the results of a bounds calculation back to the engine. The engine accumulates the bounds in the DC.

Possible error returns: none

p32_Rect is a pointer to a rectangle in device coordinates.

Bounds calculations may be done by either the engine, a simulation, or a device driver. The AccumulateBounds function is used by these three components as a means of coalescing the various bounds calculations they perform prior to the return of the data to the application by the Engine.

Parameters:

p32_Rect
A pointer to a rectangle in device coordinates.

Returns: *void* (possible error returns: none)

AllocateCurves (u32_N, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
   ULONG  u32_FuncNo;
   ULONG  u32_DcH;
   ULONG  u32_N;
};
```

Allocates u32_N storage blocks for curves. It inserts them into the doubly linked list of curves of the current path.

Each drawing function simulation must write curve records into the current path when the DC is in Path accumulation mode. This function, supplied by the Path Component, allocates and links these curves into a Path object.

This is an Engine callback routine available to the device driver only.

Parameters:

u32_N The number of storage blocks required.

Returns: *ULONG FAR **

0 error ???
!=0 A long pointer to the allocated memory

ChangePolygonMode (s32_PathId, u32_Mode, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
   ULONG  u32_FuncNo;
   ULONG  u32_DcH;
   ULONG  u32_Mode;
   ULONG  s32_PathId;
};
```

This routine converts the mode of a polygon from alternate to winding or from winding to alternate.

This is an Engine callback routine available to the device driver only for use if the device driver can handle only one of the alternate or winding modes.

Parameters:

s32_PathId
: The identifier of the path. It must be > 0.

u32_Mode
: Specifies the mode as:

```
CHANGE_TO_ALT
CHANGE_TO_WINDING
```

Returns: *Boolean*

```
TBD
```

ClipLine (p32_xy, s32_n, p32_Callback, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
   ULONG   u32_FuncNo;
   ULONG   u32_DcH;
   ULONG*  p32_CallBack;
   ULONG   s32_count;
   ULONG*  p32_xy;
};
```

This function is used by the driver to pass a polyline to the engine. The engine will clip the polyline against the current clip area and clip region, and call the device driver's callback function with a set of fully clipped lines.

The parameters are the same as for PolyLine.

The callback function may be terminated at any time by returning zero to the engine.

Parameters:

p32_xy Points to an array of x,y coordinates.

s32_n The number of x,y pairs.

p32_Callback
: The address of the device driver's Callback entry point.

Returns: *TBD*

ClipConic(p32_xy, s32_n, p32_s, p32_Callback, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
    ULONG   u32_FuncNo;
    ULONG   u32_DcH;
    ULONG*  p32_CallBack;
    ULONG*  p32_s;
    ULONG   s32_count;
    ULONG*  p32_xy;
};
```

This function is used by the driver to pass a conic to the engine. The engine will clip the conic against the current clip area and clip region, and call the device driver's callback function with a set of fully clipped lines.

The parameters are the same as for PolyFilletSharp.

The callback function may be terminated at any time by returning zero to the engine.

Parameters:

p32_xy
: Points to an array of x,y coordinates.

s32_n
: The number of x,y pairs.

p32_s
: A long pointer to an array of s32_Sharpness parameters.

p32_Callback
: The address of the device driver's Callback entry point.

Returns: *TBD*

ClipRect (p32_xy, p32_Callback, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
    ULONG   u32_FuncNo;
    ULONG   u32_DcH;
    ULONG*  p32_CallBack;
    ULONG*  p32_xy;
};
```

This function is used by the driver to pass a rectangle to the engine. The engine will clip the rectangle against the current clip area and clip region, and call the device driver's callback function with a set of fully clipped lines.

The parameters are the same as for Box.

The callback function may be terminated at any time by returning zero to the engine.

Parameters:

p32_xy Points to the x,y defining the rectangle.

p32_Callback
The address of the device driver's Callback entry point.

Returns: *TBD*

ClipScans (p32_PSL1, p32_PSL2, p32_BoundingRect, p32_Callback, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
   ULONG   u32_FuncNo;
   ULONG   u32_DcH;
   ULONG*  p32_CallBack;
   ULONG*  p32_BoundingRect.
   ULONG*  p32_PSL2;
   ULONG*  p32_PSL1;
};
```

This function is used by the driver to pass a polyscanline to the engine. The engine will clip it against the clip region, and call the device driver's callback function with a set of fully clipped lines.

The parameters are the same as for PolyScanLine.

The callback function may be terminated at any time by returning zero to the engine.

Parameters:

p32_PSL1, p32_PSL2
Long pointers to the two polyshortline structures.

p32_BoundingRect
This is a rectangle which bounds the whole figure.

p32_Callback
The address of the device driver's Callback entry point.

Returns: *TBD*

ClipPoly (p32_Callback, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
   ULONG   u32_FuncNo;
   ULONG   u32_DcH;
   ULONG*  p32_CallBack;
};
```

This function is used by the driver to determine the current clip polygon.

This allows devices which support clip areas but which do not support boolean operations on clip areas to query the result of a boolean combine area operation.

The function will be called back with three different primitives.

1. SetCurrentPosition, this denotes the beginning of a subarea.
2. Polyline
3. PolyFilletSharp

Parameters:

p32_Callback
: The address of the device driver's Callback entry point.

Returns: *TBD*

ClipPathCurves (p32_Path, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
   ULONG    u32_FuncNo;
   ULONG    u32_DcH;
   ULONG*   p32_Path;
};
```

This function is called by the either Path component or the Arcs component. The function is handled by the Clipping component. A complete path is given to the Clipping component in lpPath. The Clipping component modifies the path so that all its curves and lines are clipped against the current clipping region and polygon.

The callback function may be terminated at any time by returning zero to the engine.

Parameters:

p32_Path
: The address of the path definition.

Returns: *TBD*

NotifyTransformChange (u32_Flags, p32_Data, u32_DcH, u32_FuncNo)

```
struct ARGUMENTS {
   ULONG    u32_FuncNo;
   ULONG    u32_DcH;
   ULONG*   p32_Data;
   ULONG    u32_Flags;
};
```

This function will be called by the Transforms component whenever the transformation from WORLD to DEVICE coordinates

changes. This call provides enough information to the device so that the device can optimize its calling of the Convert function, or possibly even do all point transformations itself.

Parameters:

u32_Flags

These pass information about the complexity of the 2x2 matrix made of the fxM11, fxM12, fxM21, and fxM22 components of the composite transform from WORLD to DEVICE coordinates. They also tell if there is a translation. The flags are as follows:

```
MATRIX_SIMPLE          X'00000001'   two entries are zero
MATRIX_UNITS           X'00000002'   all entries are +1 or -1
MATRIX_XY_EXCHANGE     X'00000004'   zeros are on the diagonal
MATRIX_X_NEGATE        X'00000008'   X is hit by negative
MATRIX_Y_NEGATE        X'00000010'   Y is hit by negative
MATRIX_TRANSLATION     X'00000020'   non-zero translation
```

Examples:

```
Matrix = { 1.0 , 0.0 , 0.0 , 1.0 , 0 , 0 }   =>
         Flags = X'00000003'

Matrix = { 1.0 , 0.0 , 0.0 , 1.0 , 5 , 10 }   =>
         Flags = X'00000023'

Matrix = { 0.0 , -1.0 , 1.0 , 0.0 , 17 , -5 }   =>
         Flags = X'00000037'
```

p32_Data

A pointer to the composite matrix in fixed point notation:

```
NotifyTransformData struc
    ntd_Type    USHORT   ; indicates FIXED notation
    ntd_fxM11¦    ULONG    ; FIXED point matrix elements
    ntd_fxM12¦    ULONG
    ntd_fxM21¦    ULONG
    ntd_fxM22¦    ULONG
    ntd_M41¦    ULONG    ; long translations
    ntd_M42¦    ULONG
NotifyTransformData ends
```

Returns: *TBD*

16.2 Graphics Engine Function List

16.2.1 Device Driver Interface Function List

16.2.1.1 Functions Trappable by Device Drivers

The major function handler is responsible for dispatching the minor function according to the minor function number. For each major function handler the minor functions to be dispatched are as follows. The function number (with the command bits shown as all zeroes) is given for each minor function.

OutputArc

```
00008000 GPI: SIM  GetArcParameters
00008001 GPI: SIM  SetArcParameters
00008002 GPI: SIM  Arc
00008003 GPI: SIM  FullArcInterior
00008004 GPI: SIM  FullArcBoundary
00008005 GPI: SIM  FullArcBoth
00008006 GPI: SIM  PartialArc
0000800A GPI: SIM  PolyFillet
0000800E GPI: SIM  BoxInterior
0000800F GPI: SIM  BoxBoundary
00008010 GPI: SIM  BoxBoth
00008014 GPI: SIM  PolySpline
00008015 GPI: SIM  PolyFilletSharp
00008016 GPI: SIM  CurveToShortLine
00008017 GPI: SIM  IntersectCurves
```

OutputLine (all functions are required for all device drivers)

```
00008100 GPI: DEV  PolyLine
00008101 ENG: DEV  PolyShortLine
00008103 GPI: DEV  GetCurrentPosition
00008104 GPI: DEV  SetCurrentPosition
```

OutputMarker

```
00008200 GPI: SIM  PolyMarker
```

OutputScan (all functions are required for all device drivers)

```
; 00008300 GPI: DEV  ScanLR
  00008301 ENG: DEV  PolyScanline
```

OutputFill

```
; 00008400 GPI: SIM  FloodFill
```

Bitmap (all functions are required for all device drivers)

```
  00008500 ENG: DEV  DeviceCreateBitmap
  00008501 ENG: DEV  DeviceDeleteBitmap
  00008502 ENG: DEV  DeviceSelectBitmap
  00008504 GPI: DEV  OldGetBitmapBits
  00008505 GPI: DEV  OldSetBitmapBits
  00008506 GPI: DEV  GetPel
  00008507 GPI: DEV  SetPel
  00008508 GPI: DEV  ImageData
  00008509 GPI: DEV  Bitblt
  0000850A ENG: DEV  DeviceSetCursor
  0000850B GPI: DEV  GetBitmapBits
  0000850C GPI: DEV  SetBitmapBits
```

Textout (all functions are required for all device drivers)

```
  00008900 GPI: DEV  CharStringPos
  00008902 GPI: DEV  CharRect
  00008903 GPI: DEV  CharStr
  00008904 GPI: DEV  ScrollRect
  00008905 GPI: DEV  UpdateCursor
  00008906 GPI: DEV  QueryTextBox
  00008907 GPI: DEV  QueryTextBreak
  00008908 GPI: DEV  CharString
  00008909 GPI: DEV  QueryCharPositions
```

Area

```
  00008A00 GPI: ENG  BeginArea
  00008A01 GPI: ENG  EndArea
  00008A02 ENG: ENG  AllocateCurves
  00008A03 GPI: ENG  BeginClipArea
  00008A04 GPI: ENG  EndClipArea
  00008A05 GPI: ENG  BeginStrokes
  00008A06 GPI: ENG  EndStrokes
; 00008A07 GPI: ENG  QueryAreaState
  00008A08 GPI: ENG  BeginPath
  00008A09 GPI: ENG  EndPath
  00008A0A GPI: ENG  CloseFigure
  00008A0B GPI: ENG  DeletePath
  00008A0C GPI: ENG  CombinePath
  00008A0D GPI: ENG  ModifyPath
  00008A0E GPI: ENG  StrokePath
  00008A0F GPI: ENG  DrawPath
  00008A10 GPI: ENG  SelectClipPath
  00008A11 GPI: ENG  QueryClipPath
  00008A12 GPI: ENG  QueryNumberPaths
  00008A13 GPI: ENG  ChangePolygonMode
```

Bounds (all functions are required for all device drivers)

```
00008B04 GPI: DEV  QueryCharCorr
00008B05 GPI: DEV  GetPickWindow
00008B06 GPI: DEV  SetPickWindow
```

Clip

```
00008C00 GPI: SIM  GetClipBox
00008C01 GPI: SIM  SelectClipRegion
00008C02 GPI: SIM  IntersectClipRectangle
00008C03 GPI: SIM  ExcludeClipRectangle
00008C04 GPI: SIM  OffsetClipRegion
00008C05 SIM: SIM  SetXformRect
00008C06 GPI: SIM  QueryClipRegion
00008C07 GPI: SIM  PtVisible
00008C08 GPI: SIM  RectVisible
00008C09 DEV: SIM  GetClipRects
00008C0A GPI: SIM  SelectVisRegion
00008C0B GPI: SIM  QueryVisRegion
00008C0C GPI: SIM  ComputeRegions
00008C0D DEV: ENG  ClipScans
00008C0E ENG: ENG  ClipPathCurves
```

Region

```
00008D00 GPI: SIM  GetRegionBox
00008D01 GPI: SIM  GetRegionRects
00008D02 GPI: SIM  CreateRectRegion
00008D03 GPI: SIM  DestroyRegion
00008D04 GPI: SIM  SetRectRegion
00008D05 GPI: SIM  CombineRegion
00008D06 GPI: SIM  OffsetRegion
00008D07 GPI: SIM  EqualRegion
00008D08 GPI: SIM  PtInRegion
00008D09 GPI: SIM  RectInRegion
00008D0A GPI: SIM  PaintRegion
```

Transform

```
00008E00 GPI: SIM  Convert
00008E01 GPI: SIM  GetModelXform
00008E02 GPI: SIM  SetModelXform
00008E03 GPI: SIM  GetWindowViewportXform
00008E04 GPI: SIM  SetWindowViewportXform
00008E05 GPI: SIM  QueryViewportSize
00008E06 GPI: SIM  GetGlobalViewingTransform
00008E07 GPI: SIM  SetGlobalViewingTransform
00008E08 GPI: SIM  GetGraphicsField
00008E09 GPI: SIM  SetGraphicsField
00008E0A GPI: SIM  GetPageUnits
00008E0B GPI: SIM  SetPageUnits
00008E0C GPI: SIM  GetPageWindow
```

```
00008E0D GPI: SIM  SetPageWindow
00008E0E GPI: SIM  GetPageViewport
00008E0F GPI: SIM  SetPageViewport
00008E11 GPI: SIM  GetDCOrigin
00008E12 GPI: SIM  SetDCOrigin
00008E13 GPI: SIM  GetViewingLimits
00008E14 GPI: SIM  SetViewingLimits
```

Attributes (all functions are required for all device drivers)

```
00008F00 ENG: DEV  QueryKerning
00008F01 ENG: DEV  EnableKerning
00008F02 ENG: DEV  GetPairKerningTable
00008F03 ENG: DEV  GetTrackKernTable
00008F04 ENG: DEV  SetKernTrack
00008F05 ENG: DEV  DeviceSetAttributes
00008F06 ENG: DEV  DeviceSetGlobalAttribute
00008F07 ENG: DEV  NotifyClipChange
00008F08 ENG: DEV  RealizeFont
00008F09 GPI: DEV  ErasePS
00008F0A GPI: DEV  SetStyleRatio
00008F0B ENG: DEV  GetDCCaps
00008F0C ENG: DEV  DeviceQueryFontAttributes
00008F0D ENG: DEV  DeviceQueryFonts
00008F0E SIM: DEV  NotifyTransformChange
00008F0F GPI: DEV  GetPatternOrigin
00008F10 GPI: DEV  SetPatternOrigin
00008F11 GPI: DEV  GetCodePage
00008F12 GPI: DEV  SetCodePage
00008F13 GPI: DEV  LockDevice
00008F14 GPI: DEV  UnlockDevice
00008F15 GPI: DEV  Death
00008F16 GPI: DEV  Resurrection
00008F17 GPI: DEV  QueryLineTypeGeom
00008F18 GPI: DEV  SetLineTypeGeom
00008F19 SIM: DEV  DeviceSetDCOrigin
00008F1A SIM: DEV  GetLineOrigin
00008F1B SIM: DEV  SetLineOrigin
```

Color (all functions are required for all device drivers)

```
00009000 GPI: DEV  QueryColorData
00009001 GPI: DEV  QueryLogColorTable
00009002 GPI: DEV  CreateLogColorTable
00009003 GPI: DEV  RealizeColorTable
00009004 GPI: DEV  UnrealizeColorTable
00009005 GPI: DEV  QueryRealColors
00009006 GPI: DEV  QueryNearestColor
00009007 GPI: DEV  QueryColorIndex
00009008 GPI: DEV  QueryRGBColor
```

Query (all functions are required for all device drivers)

```
00009100 GPI: DEV  QueryDeviceBitmaps
00009101 GPI: DEV  QueryDeviceCaps
00009103 GPI: DEV  Escape
00009104 GPI: DEV  QueryHardcopyCaps
```

The major function numbers 0006, 0007, and 0008 are reserved for future use.

The device driver is allowed to reject any call it does not want to handle, as long as the call is not one of the above mentioned "required" functions. The device driver must pass any call it does not handle to the default simulation for the major handler being called.

Those functions above whose major function number is in the range 00-13 and which are not listed as "Required for device drivers" are optional device driver functions. Device drivers may hook any optional function they wish.

16.2.1.2 Functions Handled by Engine or Global Simulation

```
00001402 GPI: ENG  QueryEngineVersion
00009403 GPI: ENG  GetHandle
00009404 GPI: ENG  SetHandle
00009407 GPI: ENG  ResetDC
00009408 GPI: ENG  GetProcessControl
00009409 GPI: ENG  SetProcessControl
0000940A GPI: ENG  SaveDC
0000940B GPI: ENG  RestoreDC
0000140E GPI: ENG  OpenDC
0000940F GPI: ENG  CloseDC

00009500 GPI: ENG  DeleteSetId
00009501 GPI: ENG  LoadSymbolSet
00009502 GPI: ENG  QuerySymbolSetData
00009503 GPI: ENG  QueryNumberSetIds
00009504 GPI: ENG  QuerySetIds

00009600 GPI: ENG  CreateLogicalFont
00001601 GPI: ENG  LoadFont
00001602 GPI: ENG  UnloadFont
00009603 GPI: ENG  QueryFonts
00009604 GPI: ENG  QueryFontAttributes
00009605 GPI: ENG  ValidateEngineFont
00001606 GPI: ENG  GetCodepageTable
00001607 GPI: ENG  GetRevCodeTable
00001608 GPI: ENG  QueryLogicalFont

00009700 ENG: ENG  AccumulateJournalFile
00001701 DEV: ENG  CreateJournalFile
```

```
00001702 DEV: ENG  DeleteJournalFile
00001703 DEV: ENG  StartJournalFile
00001704 DEV: ENG  StopJournalFile
00001705 DEV: ENG  PlayJournalFile
00001706 DEV: ENG  OpenJournalFile

00009800 GPI: ENG  LoadLineTypes
00009801 GPI: ENG  QueryLineTypes
00009802 GPI: ENG  ReadLineTypes

00009900 GPI: ENG  CopyDCLoadData
00009901 GPI: ENG  QueryBitmapHandle
00009902 GPI: ENG  SetBitmapID

00009A00 GPI: ENG  CreateBitmap
00001A01 GPI: ENG  DeleteBitmap
00009A02 GPI: ENG  SelectBitmap
00001A03 GPI: ENG  GetBitmapDimension
00001A04 GPI: ENG  SetBitmapDimension
00009A05 GPI: ENG  GetAttributes
00009A06 GPI: ENG  SetAttributes
00001A07 GPI: ENG  GetBitmapParameters
00009A08 GPI: ENG  SetGlobalAttribute
00009A0B GPI: ENG  ResetBounds
00009A0C GPI: ENG  GetBoundsData
00009A0D DEV: ENG  AccumulateBounds
00009A0E GPI: ENG  SetCursor
```

16.2.2 Error Definition

Graphics Error Strategy

Note: This section is subject to change as the error definitions are clarified.

1. Sufficient validation to avoid a malfunction will always be performed where possible.
2. For environment/objects/resources e.g., SymbolSets, Fonts, Bitmaps, Regions, Segments full error checking (as defined for that function) is performed in general by the component that implements the function, i.e., Engine/DDI will perform full error checking for Symbol Sets, Fonts, Bitmaps and Regions, etc., GPI will perform full error checking for Segments, etc.
3. For Query operations full error checking (as defined for that function) is performed in general by the component that implements the function.

4. For segment drawing, drawing primitives and primitive attributes in draw mode error checking is permissive i.e., it is optional whether an invalid value is defaulted or produces the specified error but essential context checking will be performed. When storing in segment store or metafiling however, full checking is performed and all defined errors must be raised.
5. For any defined error, the application will see the same error code regardless of whether the error was logged by the GPI, Engine or device driver.

Severity of Errors

Except where specified as Warning, all Gpi errors listed fall into the category of Error, Severe Error, or Unrecoverable Error as described below:

Warning
: The function detected a problem but took some remedial action which enabled the function to complete successfully.

Error
: The function detected a problem for which it could not take any sensible remedial action. The system will be able to recover from the problem, in the sense that the state of the system, with respect to the application remains the same as at the time when the function was requested, i.e., the system has not partially executed the function.

Severe Error
: The function detected a problem from which the system cannot reestablish its state, with respect to the application, at the time when that function was requested, i.e., the system has partially executed the function, and therefore necessitates the application performing some corrective activity in order to restore the system to some known state.

Unrecoverable Error
: The function detected some problem from which the system cannot reestablish its state, with respect to the application, at the time when that call was issued and it is possible that the application cannot perform some corrective action in order to restore the system to some known state, e.g., the application provides the address of the anchor block which the system discovers is apparently corrupted.

Complete GPI Error List

```
    #define GPIERR_BASE                                0x2000
#define GPIERR_ALREADY_IN_AREA GPIERR_BASE | 0x0001
#define GPIERR_ALREADY_IN_CLIP_AREA GPIERR_BASE | 0x0002
#define GPIERR_ALREADY_IN_ELEMENT GPIERR_BASE | 0x0003
#define GPIERR_ALREADY_IN_STROKES GPIERR_BASE | 0x0004
#define GPIERR_AREA_CONTEXT_ERROR GPIERR_BASE | 0x0005
#define GPIERR_AREA_INCOMPLETE GPIERR_BASE | 0x0006
#define GPIERR_BITMAP_AND_DC_NOT_COMPAT GPIERR_BASE | 0x0007
#define GPIERR_BITMAP_IS_SELECTED GPIERR_BASE | 0x0008
#define GPIERR_BITMAP_NOT_FOUND GPIERR_BASE | 0x0009
#define GPIERR_BOX_RADIUS_TOO_LARGE GPIERR_BASE | 0x000A
#define GPIERR_CALLED_SEG_IS_CURRENT GPIERR_BASE | 0x000B
#define GPIERR_CALLED_SEG_NOT_FOUND GPIERR_BASE | 0x000C
#define GPIERR_CANNOT_DELETE_ALL_DATA GPIERR_BASE | 0x000D
#define GPIERR_CENTER_OUTSIDE_PAGE GPIERR_BASE | 0x000E
#define GPIERR_CLIP_AREA_CONTEXT_ERROR GPIERR_BASE | 0x000F
#define GPIERR_CLIP_AREA_INCOMPLETE GPIERR_BASE | 0x0010
#define GPIERR_COORDINATE_OVERFLOW GPIERR_BASE | 0x0011
#define GPIERR_CORR_FORMAT_MISMATCH GPIERR_BASE | 0x0012
#define GPIERR_DATA_TOO_LONG GPIERR_BASE | 0x0013
#define GPIERR_DC_ALREADY_ASSOCIATED GPIERR_BASE | 0x0014
#define GPIERR_DOS_ERROR GPIERR_BASE | 0x0015
#define GPIERR_DYNAMIC_SEG_SEQ_ERROR GPIERR_BASE | 0x0016
#define GPIERR_ELEMENT_CONTEXT_ERROR GPIERR_BASE | 0x0017
#define GPIERR_END_AREA_IGNORED GPIERR_BASE | 0x0018
#define GPIERR_END_AREA_WRONG_SEG GPIERR_BASE | 0x0019
#define GPIERR_END_CLIP_AREA_IGNORED GPIERR_BASE | 0x001A
#define GPIERR_END_CLIP_AREA_WRONG_SEG GPIERR_BASE | 0x001B
#define GPIERR_END_ELEMENT_IGNORED GPIERR_BASE | 0x001C
#define GPIERR_END_IMAGE_IGNORED GPIERR_BASE | 0x001D
```

```
#define GPIERR_END_IMAGE_WRONG_SEG GPIERR_BASE | 0x001E
#define GPIERR_END_PROLOG_IGNORED GPIERR_BASE | 0x001F
#define GPIERR_END_STROKES_IGNORED GPIERR_BASE | 0x0020
#define GPIERR_END_STROKES_WRONG_SEG GPIERR_BASE | 0x0021
#define GPIERR_END_SYMBOL_IGNORED GPIERR_BASE | 0x0022
#define GPIERR_ESC_CODE_NOT_SUPPORTED GPIERR_BASE | 0x0023
#define GPIERR_EXCEEDS_MAX_SEG_LENGTH GPIERR_BASE | 0x0024
#define GPIERR_FONT_NOT_FOUND GPIERR_BASE | 0x0025
#define GPIERR_FONT_NOT_LOADED GPIERR_BASE | 0x0026
#define GPIERR_GPI_BUSY GPIERR_BASE | 0x0027
#define GPIERR_GRAPHICS_SEG_IS_CURRENT GPIERR_BASE | 0x0028
#define GPIERR_IMAGE_CONTEXT_ERROR GPIERR_BASE | 0x0029
#define GPIERR_IMAGE_INCOMPLETE GPIERR_BASE | 0x002A
#define GPIERR_INCOMPATIBLE_METAFILE GPIERR_BASE | 0x002B
#define GPIERR_INCOMPLETE_LINE_DEFN GPIERR_BASE | 0x002C
#define GPIERR_INCOMPLETE_SYMBOL_DEFN GPIERR_BASE | 0x002D
#define GPIERR_INSUFFIENT_MEMORY GPIERR_BASE | 0x002E
#define GPIERR_INV_ANGLE_PARAMETER GPIERR_BASE | 0x002F
#define GPIERR_INV_ARC_CONTROL GPIERR_BASE | 0x0030
#define GPIERR_INV_ARC_PARAMETER GPIERR_BASE | 0x0031
#define GPIERR_INV_AREA_CONTROL GPIERR_BASE | 0x0032
#define GPIERR_INV_ARRAY_COUNT GPIERR_BASE | 0x0033
#define GPIERR_INV_ATTR_COUNT GPIERR_BASE | 0x0034
#define GPIERR_INV_ATTR_MODE GPIERR_BASE | 0x0035
#define GPIERR_INV_BACKGROUND_COL_ATTR GPIERR_BASE | 0x0036
#define GPIERR_INV_BACKGROUND_FM_ATTR GPIERR_BASE | 0x0037
#define GPIERR_INV_BITBLT_MIX GPIERR_BASE | 0x0038
#define GPIERR_INV_BITBLT_MODE GPIERR_BASE | 0x0039
#define GPIERR_INV_BITBLT_SOURCE_OP GPIERR_BASE | 0x003A
#define GPIERR_INV_BITBLT_TARGET_OP GPIERR_BASE | 0x003B
#define GPIERR_INV_BITMAP_DIMENSION GPIERR_BASE | 0x003C
```

```
#define GPIERR_INV_BITMAP_SHANDLE GPIERR_BASE | 0x003D
#define GPIERR_INV_BITMAP_PARAMETER GPIERR_BASE | 0x003E
#define GPIERR_INV_BOUNDARY_COLOR GPIERR_BASE | 0x003F
#define GPIERR_INV_BOX_CONTROL GPIERR_BASE | 0x0040
#define GPIERR_INV_CHAR_ANGLE_ATTR GPIERR_BASE | 0x0041
#define GPIERR_INV_CHAR_BOX_ATTR GPIERR_BASE | 0x0042
#define GPIERR_INV_CHAR_BRK_EXTRA_ATTR GPIERR_BASE | 0x0043
#define GPIERR_INV_CHAR_DIRECTION_ATTR GPIERR_BASE | 0x0044
#define GPIERR_INV_CHAR_EXTRA_ATTR GPIERR_BASE | 0x0045
#define GPIERR_INV_CHAR_MODE_ATTR GPIERR_BASE | 0x0046
#define GPIERR_INV_CHAR_SET_ATTR GPIERR_BASE | 0x0047
#define GPIERR_INV_CHAR_SHEAR_ATTR GPIERR_BASE | 0x0048
#define GPIERR_INV_CHAR_SPACING_ATTR GPIERR_BASE | 0x0049
#define GPIERR_INV_CHAR_STRING_LENGTH GPIERR_BASE | 0x004A
#define GPIERR_INV_CHAR_STRING_OPTIONS GPIERR_BASE | 0x004B
#define GPIERR_INV_CLIP_AREA_CONTROL GPIERR_BASE | 0x004C
#define GPIERR_INV_CLIP_REGION_OP GPIERR_BASE | 0x004D
#define GPIERR_INV_CODEPAGE GPIERR_BASE | 0x004E
#define GPIERR_INV_COLOR_ATTR GPIERR_BASE | 0x004F
#define GPIERR_INV_COLOR_DATA GPIERR_BASE | 0x0050
#define GPIERR_INV_COLOR_FORMAT GPIERR_BASE | 0x0051
#define GPIERR_INV_COLOR_INDEX GPIERR_BASE | 0x0052
#define GPIERR_INV_COLOR_OPTIONS GPIERR_BASE | 0x0053
#define GPIERR_INV_COMMENT_LENGTH GPIERR_BASE | 0x0054
#define GPIERR_INV_COORDINATE GPIERR_BASE | 0x0055
#define GPIERR_INV_CORR_APERTURE_TYPE GPIERR_BASE | 0x0056
#define GPIERR_INV_CORRELATE_DEPTH GPIERR_BASE | 0x0057
#define GPIERR_INV_CORRELATE_TYPE GPIERR_BASE | 0x0058
#define GPIERR_INV_DC_SHANDLE GPIERR_BASE | 0x0059
#define GPIERR_INV_DC_TYPE GPIERR_BASE | 0x005A
#define GPIERR_INV_DDA_TYPE GPIERR_BASE | 0x005B
```

```
#define GPIERR_INV_DRAW_CONTROL GPIERR_BASE | 0x005C
#define GPIERR_INV_DRAW_VALUE GPIERR_BASE | 0x005D
#define GPIERR_INV_DRAWING_MODE GPIERR_BASE | 0x005E
#define GPIERR_INV_EDIT_MODE GPIERR_BASE | 0x005F
#define GPIERR_INV_ELEMENT_NUMBER GPIERR_BASE | 0x0060
#define GPIERR_INV_EXTENDED_HEADER_LEN GPIERR_BASE | 0x0061
#define GPIERR_INV_EXTENT GPIERR_BASE | 0x0062
#define GPIERR_INV_FACENAME GPIERR_BASE | 0x0063
#define GPIERR_INV_FGBG_FM_COMBINATION GPIERR_BASE | 0x0064
#define GPIERR_INV_FIRST_CHAR GPIERR_BASE | 0x0065
#define GPIERR_INV_FONT_ATTRS GPIERR_BASE | 0x0066
#define GPIERR_INV_FONT_DEFN GPIERR_BASE | 0x0067
#define GPIERR_INV_FONT_FILENAME GPIERR_BASE | 0x0068
#define GPIERR_INV_GEOM_LINE_TYPE_ATTR GPIERR_BASE | 0x0069
#define GPIERR_INV_GEOM_LINE_WIDTH_ATTR GPIERR_BASE | 0x006A
#define GPIERR_INV_PS_SHANDLE GPIERR_BASE | 0x006B
#define GPIERR_INV_GRAPHICS_FIELD_DIM GPIERR_BASE | 0x006C
#define GPIERR_INV_GRAPHICS_FIELD_ORG GPIERR_BASE | 0x006D
#define GPIERR_INV_ID GPIERR_BASE | 0x006E
#define GPIERR_INV_IMAGE_DATA_LENGTH GPIERR_BASE | 0x006F
#define GPIERR_INV_IMAGE_DIMENSION GPIERR_BASE | 0x0070
#define GPIERR_INV_IMAGE_FORMAT GPIERR_BASE | 0x0071
#define GPIERR_INV_IMPLICIT_DRAW_FN GPIERR_BASE | 0x0072
#define GPIERR_INV_IN_IMAGE_DEFN GPIERR_BASE | 0x0073
#define GPIERR_INV_IN_ROOT_SEG_PROLOG GPIERR_BASE | 0x0074
#define GPIERR_INV_IN_VECTOR_SYMBOL GPIERR_BASE | 0x0075
#define GPIERR_INV_INFO_TABLE GPIERR_BASE | 0x0076
#define GPIERR_INV_KERNING_FLAGS GPIERR_BASE | 0x0077
#define GPIERR_INV_LENGTH GPIERR_BASE | 0x0078
#define GPIERR_INV_LINE_END_ATTR GPIERR_BASE | 0x0079
#define GPIERR_INV_LINE_JOIN_ATTR GPIERR_BASE | 0x007A
```

```
#define GPIERR_INV_LINE_PAT_SET_ATTR GPIERR_BASE | 0x007B
#define GPIERR_INV_LINE_PATTERN_ATTR GPIERR_BASE | 0x007C
#define GPIERR_INV_LINE_TYPE_ATTR GPIERR_BASE | 0x007D
#define GPIERR_INV_LINE_TYPE_CODEPOINT GPIERR_BASE | 0x007E
#define GPIERR_INV_LINE_TYPE_TABLE_ID GPIERR_BASE | 0x007F
#define GPIERR_INV_LINE_WIDTH_ATTR GPIERR_BASE | 0x0080
#define GPIERR_INV_MARKER_ATTR GPIERR_BASE | 0x0081
#define GPIERR_INV_MARKER_BOX_ATTR GPIERR_BASE | 0x0082
#define GPIERR_INV_MARKER_SET_ATTR GPIERR_BASE | 0x0083
#define GPIERR_INV_METAFILE SHANDLE GPIERR_BASE | 0x0084
#define GPIERR_INV_METAFILE_FILENAME GPIERR_BASE | 0x0085
#define GPIERR_INV_METAFILE_FUNCTION GPIERR_BASE | 0x0086
#define GPIERR_INV_METAFILE_LENGTH GPIERR_BASE | 0x0087
#define GPIERR_INV_METAFILE_OFFSET GPIERR_BASE | 0x0088
#define GPIERR_INV_MICROPS_FUNCTION GPIERR_BASE | 0x0089
#define GPIERR_INV_FM_ATTR GPIERR_BASE | 0x008A
#define GPIERR_INV_MULTIPLIER GPIERR_BASE | 0x008B
#define GPIERR_INV_NAME GPIERR_BASE | 0x008C
#define GPIERR_INV_NO_IMAGE_DATA_ORDERS GPIERR_BASE | 0x008D
#define GPIERR_INV_OFFSET GPIERR_BASE | 0x008E
#define GPIERR_INV_OR_INCOMPAT_OPTIONS GPIERR_BASE | 0x008F
#define GPIERR_INV_OR_INCONSISTENT_LENS GPIERR_BASE | 0x0090
#define GPIERR_INV_ORDER GPIERR_BASE | 0x0091
#define GPIERR_INV_ORDER_LENGTH GPIERR_BASE | 0x0092
#define GPIERR_INV_ORDER_OUTSIDE_PROLOG GPIERR_BASE | 0x0093
#define GPIERR_INV_ORDERING_PARAMETER GPIERR_BASE | 0x0094
#define GPIERR_INV_PATTERN_ATTR GPIERR_BASE | 0x0095
#define GPIERR_INV_PATTERN_SET_ATTR GPIERR_BASE | 0x0096
#define GPIERR_INV_PICK_APERTURE_DIM GPIERR_BASE | 0x0097
#define GPIERR_INV_PICK_APERTURE_OPTION GPIERR_BASE | 0x0098
#define GPIERR_INV_PICK_NUMBER GPIERR_BASE | 0x0099
```

```
#define GPIERR_INV_PLAY_METAFILE_OPTION GPIERR_BASE | 0x009A
#define GPIERR_INV_PORT_NAME GPIERR_BASE | 0x009B
#define GPIERR_INV_POSITIONING_VALUE GPIERR_BASE | 0x009C
#define GPIERR_INV_PRIMITIVE_TYPE GPIERR_BASE | 0x009D
#define GPIERR_INV_PS_DIMENSION GPIERR_BASE | 0x009E
#define GPIERR_INV_REGION_CONTROL GPIERR_BASE | 0x009F
#define GPIERR_INV_REGION_SHANDLE GPIERR_BASE | 0x00A0
#define GPIERR_INV_REGION_MIX GPIERR_BASE | 0x00A1
#define GPIERR_INV_RESERVED_FIELD GPIERR_BASE | 0x00A2
#define GPIERR_INV_RESET_OPTIONS GPIERR_BASE | 0x00A3
#define GPIERR_INV_RESOURCE_SHANDLE GPIERR_BASE | 0x00A4
#define GPIERR_INV_RESOURCE_ID GPIERR_BASE | 0x00A5
#define GPIERR_INV_RGBCOLOR GPIERR_BASE | 0x00A6
#define GPIERR_INV_ROUNDING_PARAMETERS  GPIERR_BASE | 0x00A7
#define GPIERR_INV_SCAN_COUNT GPIERR_BASE | 0x00A8
#define GPIERR_INV_SEG_ATTR GPIERR_BASE | 0x00A9
#define GPIERR_INV_SEG_ATTR_CODE GPIERR_BASE | 0x00AA
#define GPIERR_INV_SEG_CH_LENGTH GPIERR_BASE | 0x00AB
#define GPIERR_INV_SEG_NAME GPIERR_BASE | 0x00AC
#define GPIERR_INV_SET_ID GPIERR_BASE | 0x00AD
#define GPIERR_INV_SHARPNESS_PARAMETER GPIERR_BASE | 0x00AE
#define GPIERR_INV_SOURCE_PS_SHANDLE GPIERR_BASE | 0x00AF
#define GPIERR_INV_SOURCE_OFFSET GPIERR_BASE | 0x00B0
#define GPIERR_INV_SRC_OR_TARG GPIERR_BASE | 0x00B1
#define GPIERR_INV_START GPIERR_BASE | 0x00B2
#define GPIERR_INV_START_SCAN GPIERR_BASE | 0x00B3
#define GPIERR_INV_STOP_DRAW_VALUE GPIERR_BASE | 0x00B4
#define GPIERR_INV_SYMBOL_SET_CODEPOINT GPIERR_BASE | 0x00B5
#define GPIERR_INV_SYMBOL_SET_FORMAT GPIERR_BASE | 0x00B6
#define GPIERR_INV_SYMBOL_SET_LENGTHGPIERR_BASE | 0x00B7
#define GPIERR_INV_SYMBOL_SET_OPTION GPIERR_BASE | 0x00B8
```

```
#define GPIERR_INV_SYMBOL_SET_TYPE GPIERR_BASE | 0x00B9
#define GPIERR_INV_TEXT_ALIGN_ATTR GPIERR_BASE | 0x00BA
#define GPIERR_INV_TRANSFORM_PARAMETER  GPIERR_BASE | 0x00BB
#define GPIERR_INV_TRANSFORM_TYPE GPIERR_BASE | 0x00BC
#define GPIERR_INV_UNITS GPIERR_BASE | 0x00BD
#define GPIERR_INV_USAGE GPIERR_BASE | 0x00BE
#define GPIERR_INV_VIEW_LIMIT_SPEC GPIERR_BASE | 0x00BF
#define GPIERR_INV_WINDOW_SPECIFICATION GPIERR_BASE | 0x00C0
#define GPIERR_LABEL_NOT_FOUND GPIERR_BASE | 0x00C1
#define GPIERR_LOGICAL_FONT_NOT_FOUND GPIERR_BASE | 0x00C2
#define GPIERR_MATRIX_OVERFLOW GPIERR_BASE | 0x00C3
#define GPIERR_MET_FILENAME_NOT_FOUND GPIERR_BASE | 0x00C4
#define GPIERR_NAMED_SEG_NOT_CHAINED GPIERR_BASE | 0x00C5
#define GPIERR_NO_BEGIN_IMAGE_ORDER GPIERR_BASE | 0x00C6
#define GPIERR_NO_BITMAP_SELECTED_IN_DC GPIERR_BASE | 0x00C7
#define GPIERR_NO_CURRENT_ELEMENT GPIERR_BASE | 0x00C8
#define GPIERR_NO_CURRENT_GRAPHICS_SEG GPIERR_BASE | 0x00C9
#define GPIERR_NOT_IN_IMAGE_BRACKET GPIERR_BASE | 0x00CA
#define GPIERR_NOT_IN_STORE_MODE GPIERR_BASE | 0x00CB
#define GPIERR_OVERRAN_SEG GPIERR_BASE | 0x00CC
#define GPIERR_PRIMITIVE_STACK_EMPTY GPIERR_BASE | 0x00CD
#define GPIERR_PROLOG_ERROR GPIERR_BASE | 0x00CE
#define GPIERR_PROLOG_SEG_ATTR_NOT_SET GPIERR_BASE | 0x00CF
#define GPIERR_PS_ALREADY_ASSOCIATED GPIERR_BASE | 0x00D0
#define GPIERR_REFSEG_NOT_CHAINED GPIERR_BASE | 0x00D1
#define GPIERR_REFSEG_UNKNOWN GPIERR_BASE | 0x00D2
#define GPIERR_REL_LINE_OUTSIDE_WCS GPIERR_BASE | 0x00D3
#define GPIERR_RESERVED_FIELD_NOT_ZERO GPIERR_BASE | 0x00D4
#define GPIERR_RESOURCE_DEPLETION GPIERR_BASE | 0x00D5
#define GPIERR_RIGHT_LESS_THAN_LEFT GPIERR_BASE | 0x00D6
#define GPIERR_SEG_AND_REFSEG_ARE_SAME GPIERR_BASE | 0x00D7
```

```
#define GPIERR_SEG_CALL_RECURSIVE GPIERR_BASE | 0x00D8
#define GPIERR_SEG_CALL_STACK_EMPTY GPIERR_BASE | 0x00D9
#define GPIERR_SEG_CALL_STACK_FULL GPIERR_BASE | 0x00DA
#define GPIERR_SEG_CONTEXT_ERROR GPIERR_BASE | 0x00DB
#define GPIERR_SEG_UNKNOWN GPIERR_BASE | 0x00DC
#define GPIERR_SET_ID_ALREADY_IN_USE GPIERR_BASE | 0x00DD
#define GPIERR_SET_ID_NOT_A_SYMBOL_SET GPIERR_BASE | 0x00DE
#define GPIERR_SET_ID_NOT_LOADED GPIERR_BASE | 0x00DF
#define GPIERR_START_INDEX GPIERR_BASE | 0x00E0
#define GPIERR_STOP_DRAW_OCCURRED GPIERR_BASE | 0x00E1
#define GPIERR_STROKES_CONTEXT_ERROR GPIERR_BASE | 0x00E2
#define GPIERR_STROKES_INCOMPLETE GPIERR_BASE | 0x00E3
#define GPIERR_SYMBOL_SET_LEN_TOO_SMALL GPIERR_BASE | 0x00E4
#define GPIERR_SYMBOL_SET_NOT_FOUND GPIERR_BASE | 0x00E5
#define GPIERR_SYMBOL_SET_TYPE_MISMATCH GPIERR_BASE | 0x00E6
#define GPIERR_TARGET_SEG_UNKNOWN GPIERR_BASE | 0x00E7
#define GPIERR_TOKEN_NOT_ASTERISK GPIERR_BASE | 0x00E8
#define GPIERR_TOP_LESS_THAN_BOTTOM GPIERR_BASE | 0x00E9
#define GPIERR_TRUNCATED_ORDER GPIERR_BASE | 0x00EA
#define GPIERR_UNSUPPORTED_ATTRS_MASK GPIERR_BASE | 0x00EB
#define GPIERR_UNSUPPORTED_DEFS_MASK GPIERR_BASE | 0x00EC
#define GPIERR_WIDTH_OR_DEPTH_TOO_BIG GPIERR_BASE | 0x00ED
#define GPIERR_WINDOW_COLUMN_IS_INVALID GPIERR_BASE | 0x00EE
#define GPIERR_WINDOW_DEPTH_IS_INVALID GPIERR_BASE | 0x00EF
#define GPIERR_WINDOW_LIMS_OUTSIDE_PAGE GPIERR_BASE | 0x00F0
#define GPIERR_WINDOW_ROW_IS_INVALID GPIERR_BASE | 0x00F1
#define GPIERR_WINDOW_WIDTH_IS_INVALID GPIERR_BASE | 0x00F2
#define GPIERR_Z_COORDS_UNEQUAL GPIERR_BASE | 0x00F3
#define GPIERR_3D_NOT_SUPPORTED GPIERR_BASE | 0x00F4
```

Error Definition by Engine/DDI Functions

General errors applicable to many functions. The following errors are required (i.e., not permissive). They should be detected and logged in all circumstances.

GPIERR_MATRIX_OVERFLOW
All functions that may result in matrix computation.

GPIERR_INSUFFIENT_MEMORY
All functions that result in memory allocation.

GPIERR_INV_DC_SHANDLE
All functions with hdc as explicit or implicit parameter.

GPIERR_INV_BITMAP_SHANDLE
All functions with hbm as explicit or implicit parameter.

GPIERR_INV_REGION_SHANDLE
All functions with hrgn as explicit or implicit parameter.

GPIERR_INV_METAFILE SHANDLE
All functions with hmf as explicit or implicit parameter.

GPIERR_INV_COORDINATE
All functions with coordinates as parameters.

GPIERR_AREA_CONTEXT_ERROR
All functions that are invalid inside an open area bracket (to be defined by MS).

GPIERR_CLIP_AREA_CONTEXT_ERROR
All functions that are invalid inside an open clip area bracket (to be defined by MS).

GPIERR_STROKES_CONTEXT_ERROR
All functions that are invalid inside an open strokes bracket (to be defined by MS).

GPIERR_DOS_ERROR (unexpected CP/DPS error)
All functions that directly or indirectly issue MS OS/2 calls.

Engine functions with required errors:

SetPickWindow

Required errors:

```
GPIERR_WINDOW_LIMS_OUTSIDE_PAGE
```

GetAttributes

Principal errors:

```
GPIERR_INV_PRIMITIVE_TYPE
GPIERR_UNSUPPORTED_ATTRS_MASK
```

BeginArea

Required errors:

GPIERR_ALREADY_IN_AREA

BeginClipArea

Required errors:

GPIERR_ALREADY_IN_CLIP_AREA

BeginStrokes

Required errors:

GPIERR_ALREADY_IN_STROKES

DDA

Required errors:

GPIERR_INV_DDA_TYPE
GPIERR_INV_ARRAY_COUNT

EndArea

Required errors:

GPIERR_END_AREA_IGNORED

EndClipArea

Required errors:

GPIERR_END_CLIP_AREA_IGNORED

EndStrokes

Required errors:

GPIERR_END_STROKES_IGNORED

QueryDDA

Required errors:

GPIERR_INV_DDA_TYPE

QueryTextBox

Required errors:

GPIERR_INV_CHAR_STRING_LENGTH
GPIERR_INV_ARRAY_COUNT

QueryTextBreak

Required errors:

GPIERR_INV_CHAR_STRING_LENGTH
GPIERR_INV_EXTENT

CharRect

Required errors:

```
As required for MS OS/2 VIO compatibility
```

CharStr

Required errors:

```
As required for MS OS/2 VIO compatibility
```

ScrollRect

Required errors:

```
As required for MS OS/2 VIO compatibility
```

UpdateCursor

Required errors:

```
As required for MS OS/2 VIO compatibility
```

BitBlt

Required errors:

```
GPIERR_INV_BITBLT_SOURCE_OP
GPIERR_INV_BITBLT_TARGET_OP
```

CreateBitmap

Required errors:

```
GPIERR_INV_BITMAP_PARAMETER
GPIERR_INV_USAGE
```

DeleteBitmap

Required errors:

```
GPIERR_BITMAP_IS_SELECTED
```

GetBitmapBits

Required errors:

```
GPIERR_INV_SCAN_COUNT
GPIERR_INV_START_SCAN
GPIERR_INV_INFO_TABLE
GPIERR_INV_DC_TYPE
GPIERR_NO_BITMAP_SELECTED_IN_DC
```

SelectBitmap

Required errors:

```
GPIERR_BITMAP_AND_DC_INCOMP
```

SetBitmapBits

Required errors:

```
GPIERR_INV_SCAN_COUNT
GPIERR_INV_START_SCAN
GPIERR_INV_INFO_TABLE
GPIERR_INV_DC_TYPE
GPIERR_NO_BITMAP_SELECTED_IN_DC
```

CombineRegion

Required errors:

```
GPIERR_INV_REGION_MIX
GPIERR_INV_CLIP_REGION_OP
```

CreateRectRegion

Required errors:

```
GPIERR_INV_ARRAY_COUNT
```

DestroyRegion

Required errors:

```
GPIERR_INV_CLIP_REGION_OP
```

GetRegionRects

Required errors:

```
GPIERR_INV_REGION_CONTROL
```

OffsetRegion

```
GPIERR_INV_CLIP_REGION_OP
```

SetRectRegion

Required errors:

```
GPIERR_INV_ARRAY_COUNT
```

CreateLogicalFont

Required errors:

```
GPIERR_FONT_NOT_LOADED
GPIERR_INV_SET_ID
GPIERR_SET_ID_ALREADY_IN_USE
GPIERR_INV_FONT_ATTRS
GPIERR_INV_NAME
OTHERS TBD
```

DeleteSetId

Required errors:

```
GPIERR_INV_SET_ID
GPIERR_SET_ID_NOT_LOADED
```

EnableKerning

Required errors:

GPIERR_INV_KERNING_FLAGS

GetKerningPairTable

Required errors:

GPIERR_INV_ARRAY_COUNT

LoadFont

Required errors:

GPIERR_FONT_NOT_FOUND
GPIERR_INV_FONT_FILENAME
GPIERR_INV_FONT_DEFN
OTHERS TBD

LoadSymbolSet

Required errors:

GPIERR_INV_NAME
GPIERR_INV_SYMBOL_SET_LENGTH
GPIERR_INV_EXTENDED_HEADER_LEN
GPIERR_INCOMPLETE_SYMBOL_DEFN
GPIERR_INV_SET_ID
GPIERR_INV_SYMBOL_SET_TYPE
GPIERR_SYMBOL_SET_TYPE_MISMATCH
GPIERR_SET_ID_ALREADY_IN_USE
GPIERR_INV_SYMBOL_SET_FORMAT
GPIERR_SYMBOL_SET_LEN_TOO_SMALL
GPIERR_INV_SYMBOL_SET_CODEPOINT
GPIERR_INV_SYMBOL_SET_OPTION
GPIERR_INV_IN_VECTOR_SYMBOL_DEFN

QueryBitmapHandle

Required errors:

GPIERR_BITMAP_NOT_FOUND
GPIERR_INV_SET_ID

QueryFontAttributes

Required errors:

GPIERR_INV_LENGTH

QueryFonts

Required errors:

GPIERR_INV_FACENAME
GPIERR_INV_ARRAY_COUNT (METRICS)
GPIERR_INV_LENGTH (metrics_length)

QuerySetIds

Required errors:

GPIERR_INV_ARRAY_COUNT

QuerySymbolSetData

Required errors:

```
GPIERR_INV_SET_ID
GPIERR_SET_ID_NOT_A_SYMBOL_SET
GPIERR_INV_ARRAY_COUNT
```

SetBitmapID

Required errors:

```
GPIERR_INV_SET_ID
GPIERR_SET_ID_ALREADY_IN_USE
```

SetCodePage

Required errors:

```
GPIERR_INV_CODEPAGE
```

UnloadFont

Required errors:

```
GPIERR_FONT_NOT_LOADED
GPIERR_INV_FONT_FILENAME
```

OpenDc

Required errors:

```
GPIERR_INV_DC_TYPE
GPIERR_INV_LENGTH
GPIERR_TOKEN_NOT_ASTERISK
Others TBD
```

RestoreDc

Required errors:

```
GPIERR_INV_ID
```

SetGlobalViewingXform

Required errors:

```
GPIERR_INV_TRANSFORM_TYPE
GPIERR_INV_TRANSFORM_PARAMETER
```

SetPageUnits

Required errors:

```
GPIERR_INV_PS_DIMENSION
GPIERR_WIDTH_OR_DEPTH_TOO_BIG
GPIERR_INV_OR_INCOMPAT_OPTIONS (i.e., units)
```

Convert

Principal errors:

GPIERR_INV_SRC_OR_TARG
GPIERR_INV_ARRAY_COUNT

QueryViewportSize

Principal errors:

GPIERR_INV_UNITS
GPIERR_INV_ARRAY_COUNT

CreateLogColorTable

Principal errors:

GPIERR_INV_COLOR_OPTIONS
GPIERR_INV_COLOR_FORMAT
GPIERR_START_INDEX
GPIERR_INV_ARRAY_COUNT
GPIERR_INV_COLOR_DATA

QueryColorData

Principal errors:

GPIERR_INV_ARRAY_COUNT

QueryLogColorTable

Principal errors:

GPIERR_INV_COLOR_OPTIONS
GPIERR_START_INDEX
GPIERR_INV_ARRAY_COUNT

QueryRealColors

Principal errors:

GPIERR_INV_COLOR_OPTIONS
GPIERR_START_INDEX
GPIERR_INV_ARRAY_COUNT

QueryNearestColor

Principal errors:

GPIERR_INV_COLOR_OPTIONS
GPIERR_INV_RGBCOLOR

QueryColorIndex

Principal errors:

GPIERR_INV_COLOR_OPTIONS
GPIERR_INV_RGBCOLOR

QueryRGBColor

Principal errors:

GPIERR_INV_COLOR_OPTIONS
GPIERR_INV_COLOR_INDEX

LoadLineTypes

Principal errors:

```
GPIERR_INV_OR INCONSISTENT_LENS
GPIERR_INV_LINE_TYPE_CODEPOINT
GPIERR_INV_RESERVED_FIELD
GPIERR_INV_LTTID
GPIERR_INCOMPLETE_LINE_DEFN
```

QueryLineTypes

Principal errors:

```
GPIERR_INV_LENGTH
```

QueryDeviceBitmaps

Principal errors:

```
GPIERR_INV_ARRAY_COUNT
```

QueryDeviceCaps

Principal errors:

```
GPIERR_INV_ARRAY_COUNT
GPIERR_INV_ELEMENT_NUMBER
```

Escape

Principal errors:

```
GPIERR_INV_LENGTH
GPIERR_ESC_CODE_NOT_SUPPORTED
```

16.2.3 Standard Default Values

16.2.3.1 Device Independent Values

Variable	Var Type	Hex Value	Description
arc parameter p	s32	'00000001'X	
arc parameter q	s32	'00000001'X	
arc parameter r	s32	'00000000'X	
arc parameter s	s32	'00000000'X	
bounds xmin	s32	'7FFFFFFF'X	
bounds ymin	s32	'7FFFFFFF'X	
bounds xmax	s32	'80000000'X	
bounds ymax	s32	'80000000'X	
character angle x	s32	'0001'X	No rotation.
y	s32	'0000'X	
character direction	u16	'0001'X	Left to right.
character spacing			Standard spacing.
width multiplier		'00000000'X	

```
     height multiplier                  '00000000'X
character precision        u16          '0001'X          Precision 1.
character set              u16          '00F0'X          Base symbol set.
character shear x          s32          '0000'X          No shear.
                 y         s32          '0001'X
text alignment horizontal  s16          'FFFF'X          Standard alignment.
               vertical    s16          'FFFF'X
character break extra
          code point       u16          '0000'X
          width                         '00000000'X      Standard spacing.
          height                        '00000000'X
character extra
          width                         '00000000'X      Standard spacing.
          height                        '00000000'X

current position x         s32          '00000000'X
                 y         s32          '00000000'X

line end                   u16          '0001'X          Flat.
line geometric width       s32          '00000001'X      1
line join                  u16          '0001'X          Bevel.
line pattern set           u16          '00F0'X          Base pattern set.
line pattern symbol        u16          '0010'X          Solid shading.
line type                  u16          '0007'X          Solid.
line width                 u32          '00010000'X      Normal.

marker set                 u16          '00F0'X          Base marker set.
marker symbol              u16          '0001'X          Cross.


Variable                 Var Type    Hex Value       Description
~~~~~~~~                 ~~~~~~~~    ~~~~~~~~~       ~~~~~~~~~~~

character, line, marker, pattern and image:
color - foreground         u32          '00000007'X      screen: white
                                                         printer: black
character, line, marker, pattern and image:
color - background         u32          '00000000'X      screen: black
                                                         printer: white
character, line, marker, pattern and image:
mix - foreground           u16          '0002'X          Overpaint.

character, line, marker, pattern and image:
mix - background           u16          '0005'X          Transparent.


pattern origin x           s32          '00000000'X
               y           s32          '00000000'X
pattern set                u16          '00F0'X          Base pattern set.
pattern symbol             u16          '0010'X          Solid shading.


viewing limits xleft       s32          'FFFF8000'X      -32K.
viewing limits ybottom     s32          'FFFF8000'X      -32K.
viewing limits xright      s32          '00007FFF'X      +32K.
viewing limits ytop        s32          '00007FFF'X      +32K.


ProcessControlFlags        u32:
Bit 0       draw                            '1'B
Bit 1       bounds                          '0'B
Bit 2       correlate                       '0'B
Bits 3-31   reserved                        '0'B
```

ModelTransform	Unity (See Note 2.)
WindowViewportTransform	Unity (See Note 2.)
GlobalViewingTransform	Unity (See Note 2.)
PageUnits	PelsUp
PageSize	Device Size in pels (i.e., maximized window, paper size, etc.)
Graphics Field	Determined by PageUnits, PageSize (see SetPageUnits)
PageWindow	Determined by PageUnits, PageSize (see SetPageUnits)
PageViewport	Determined by PageUnits, PageSize (see SetPageUnits)
Device Transform	Determined by PageWindow, PageViewport.

Note: 1. Reset will never modify PageUnits or PageSize.

Note: 2. Unity Transformation Matrix:

element fxM11	'00010000'X	element 1,1 = 1
element fxM12	'00000000'X	element 1,2 = 0
element fxM21	'00000000'X	element 2,1 = 0
element fxM22	'00010000'X	element 2,2 = 1
element M41	'00000000'X	element 4,1 = 0
element M42	'00000000'X	element 4,2 = 0

16.2.3.2 Device Dependent Values

Variable	Var Type	Device	Hex Value	Description
pick window xmin	s32		-a/2	
pick window ymin	s32		-b/2	
pick window xmax	s32		+a/2	
pick window ymax	s32		+b/2	

where a and b are the x and y dimensions of a rectangle in page coordinate space that produces a square in device coordinates with both dimensions equal to the default character cell height.

character cell width	s32	Device dependent. Normal default cell
height	s32	size as returned by QueryDeviceCaps.
marker cell width	s32	Device dependent. Marker cell size as
height	s32	returned by QueryDeviceCaps.