Microsoft® Operating System/2

Windows Presentation Manager Reference

Volume 3

Microsoft Corporation

Document Number 07-01-87-003
Part Number 00256

Contents

Figures

Preface

The *Microsoft Operating System/2 Windows Presentation Manager Reference*, Volumes 1, 2, and 3, is derived from the latest draft of the functional specification of the Windows Presentation Manager. Although this documentation does not represent the final Windows 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 Windows Presentation Manager described in this document are subject to change. It is strongly recommended that the documentation be read for informational purposes only.

Chapter 15
Tool Kit Utilities

15.1 DIALOG BOX EDITOR USER SPECIFICATION

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

The 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 a 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 included in the resource definition file with the rcinclude directive. These files are processed by the 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, Presentation Manager automatically sends a WM_COMMAND message with ID Value 1 while when the ESCAPE key is pressed, Presentation Manager sends a WM_COMMAND message with an ID Value of 2. The default ID Symbols for the ID Values (in windows.h) are IDOK for 1 and IDCANCEL for 2.

The following diagrams illustrate how the various files work with various programs.

The .h, .res, and .dlg fit together with the Dialog Box Editor thus:

The .rc, .dlg, .h, and .res files fit together thus:

The .res, .h, and source files fit together thus:

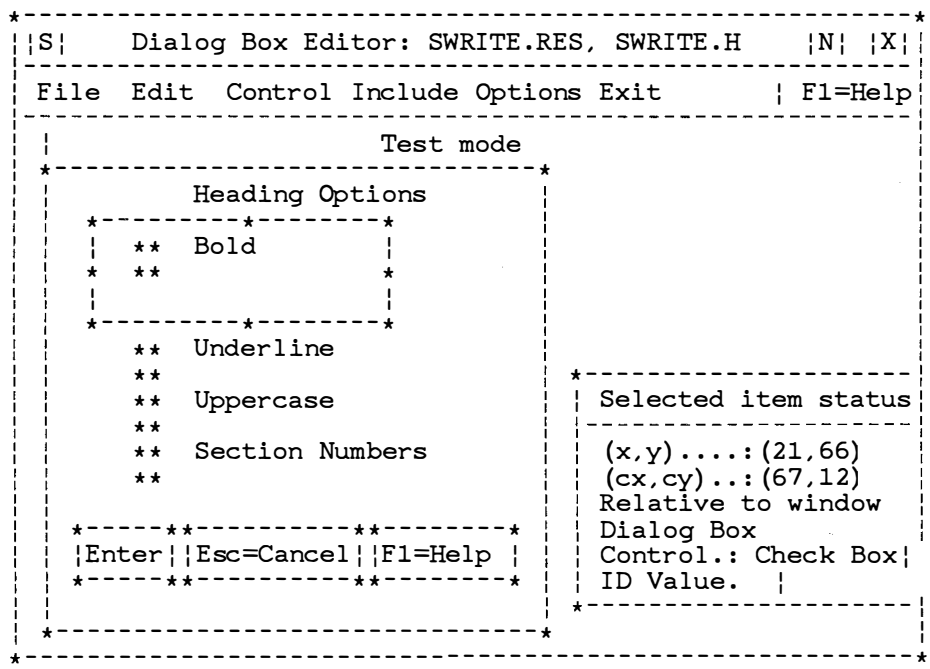
15.1.2 APPLICATION APPEARANCE

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 Presentation Manager. There are also some inaccuracies introduced by putting the image in a Write document.



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 as explained in the section, "Resizing the Main Window".

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. A picture of the window is below.

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". See section "Position Relative to..." in the section, "Edit PopDown".

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 (as described in the section, "Dialog Box Manipulations" above) 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 **x** it, and
- F1=Help.

The underlined 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, change 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:

- **N** ew,
- **O** pen...,
- **S** ave,
- Save **A** s..., and
- **V** iew 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 (see the section, "Open...") 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 (seethe section, "View 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 (see the section "Save As...")

```
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:

- **N** ew,
- **O** pen...,
- **S** ave,
- Save **A** s...,
- **V** iew Include...,
- **H** ex Mode.

This pop-down deals directly with the include files (.h files), providing a way to change the include file being edited without changing 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. See section "Open" for the standard Open File dialog box.

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... (Section "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...,
- C **l** ear Control/Dialog Box,
- **N** ew Dialog Box...,
- Rena **m** e Dialog Box...,
- Positio **n** relative to...,
- **S** tyles...,
- **G** roup Move,
- **D** uplicate Control,
- Resource Proper **t** ies...

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 (see the section, “Duplicate Control”)

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. (see the section, "New 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 (see the section, "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 shown in the section, "New Dialog Box..." above, 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), and
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, t hen 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 anytime 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 buttons 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 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 ic on to use. For all controls, you my 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 (see the sections, "Control Pop-Down", "Control Duplicate", and "Duplicating a 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 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 IDOK).

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 which 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. They 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 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 receive s 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 Presentation Manager to notify the application whenever the user clicks or double clicks on an item in the list box. The Sort option ca uses 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 displa ying 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

The only thing that can be set for a Scroll Bar is the ID field is standard. 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 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 (see the section, "Moving a Group of Controls"). 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 (see the section, "Duplicating a Control"). 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 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:

- **C** heck Box,
- **R** adio Button,
- **P** ush Button,
- **G** roup Box,
- **H** orz. Scroll,
- **V** ert. Scroll,
- **L** ist Box,
- **E** dit,
- **T** ext,
- **F** rame,
- **R** ect,
- **I** con.

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, see the section, “Styles...”.

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 (see the section, "Styles..."), 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 (see the section, "Group Move") 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, rel ease 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 (see the section, "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:

- **T** est Mode,
- T **r** anslate Mode,
- **G** rid...,
- 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. Transla te 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 translating the application to another language. All strings in the string table resource must also be translated and any static text control receiving t hose strings must be large enough for the translated text.

15.1.4.6.3 Grid...

The Grid choice puts up 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 tr y 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. This has nothing to do with the section, "Group Move").

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, as shown in the preceding Figure). 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:

- **E** xit Dialog Box Editor
- **C** ontinue 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, the F1 key, will invoke user interface help for the Dialog Box Editor as described in 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, a Presentation Manager application. It describes the physical appearance of the application when running under Presentation Manager, and also how the user interacts with the application, ie., what the assorted commands do, and how to edit font characters.

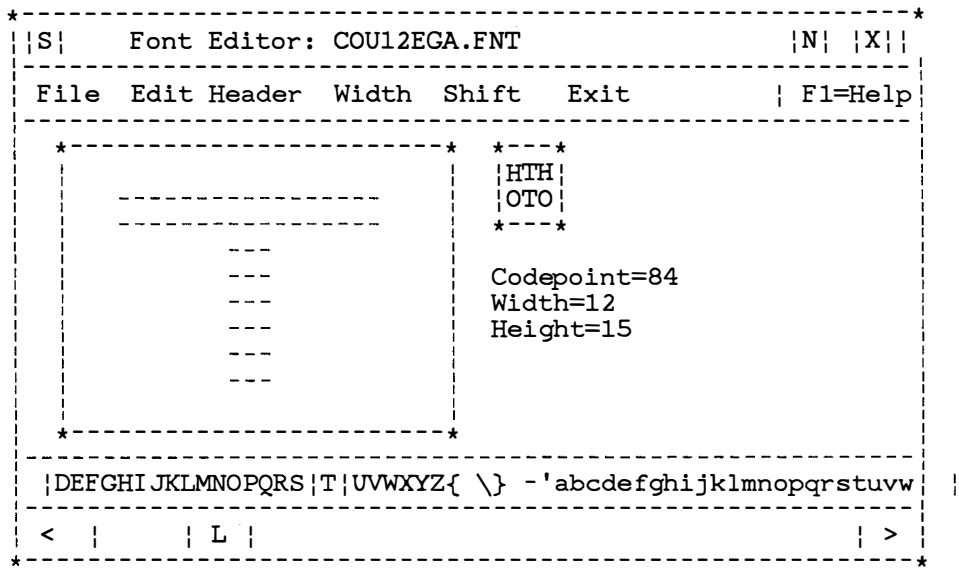
The Font Editor lets the user edit font files to use with applications. A font file consists of a header and a collection of character bitmaps that represent the individual letters, digits, and punctuation characters that can be used to display text on a display device. Application writers who want to use fonts in their applications must add the new font files to a font resource file. *Note:* This specification uses the definition of fonts from Appendix A. 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, ie., 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, described in the section, "Application Action Bar", the pointer will appear as a horizontal bar. When the Font Editor is in Add/Delete Column mode, described in the section, "Application Action Bar", 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. 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.

15.2.2 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:

- **F** ile,
- **E** dit,
- **H** eader,
- **W** idth,
- **S** hift, and
- E **x** it.

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:

- **N** ew,

- **O** pen...,
- **S** ave, and
- Save **A** s....

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:

- **C** ut,
- **C** opy,
- **P** aste,
- **U** ndo, and
- **R** estore.

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:

- **N** aming...,
- **G** eneral...,
- **S** izes..., and
- **R** elations....

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 seperate dialog boxes.

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

Typeface name
: (entry field) The typeface name to which the font is designed, eg. 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, and
- 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 - eg. 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 - ie. 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 uppercase characters.

'x' Height
: (entry field) The (average) height above the baseline for lowercase 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, and
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, and
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 Presentation Manager application. It describes the physical appearance of the application when running under Presentation Manager, and also how the user interacts with the application, ie., 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 refer to "CGA" compatible displays (640x200). Med-res refer to "EGA/VGA" compatible displays (640x350, 640x480). Hi-res refer to any displays which have a higher resolution than the "EGA/VGA" displays. Also, the following dimensions given for icons and pointers are still subject to change depending on what 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 h and found in some message boxes. Pointers are used by 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). 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	0	1	1
XOR mask:	0	1	0	1

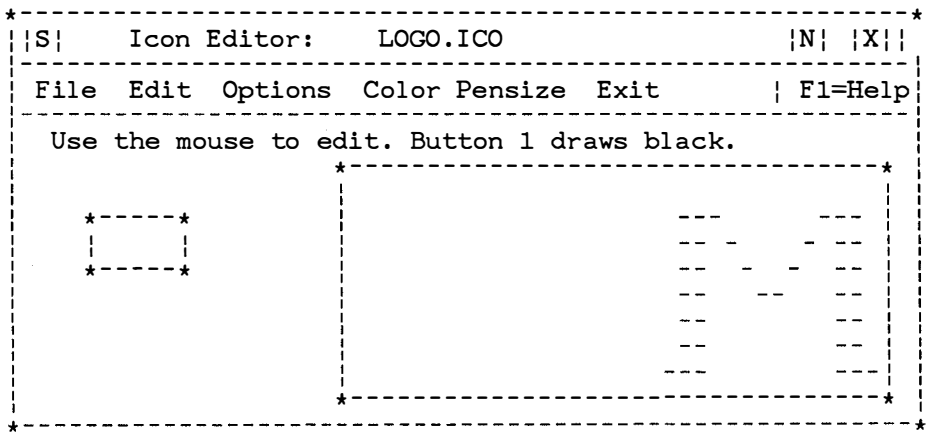
Bitmaps contain anywhere from 1x1 to 99x99 pixels. Their size is defined by the user while using the Icon Editor. They can only contain two kinds of pixels in them: black pixels or white pixels.

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 sc
"Use the mouse to Edit.  Button1 toggles bl/wh.  Button2 toggl
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, ie., 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, described in the section, "Application Action Bar", the pointer will appear as a bullseye. When the Icon Editor is in Select mode, described in the section, "Application Action Bar", 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, described in the section, "Application Action Bar". 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 mo use button and dragging the pointer. The user can also draw straight lines by selecting the "Draw straight" choice from the Options pop-down, as described in the section, “Application Action Bar”.

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 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 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. 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 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, **Edit**, **O** ptions, **C** olor, **P** ensize, and **E** xit. The underlined character is the mnemonic for the choice. There will also be the choice F1=Help in the rightmost position.

15.3.2.2.1 File Pop-down

The File pop-down has four choices:

- **N** ew
- **O** pen...
- **S** ave
- Save **A** s...

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:

- **S** elect
- Select **A** ll
- **C** lear
- C **u** t
- Cop **y**
- **P** aste
- **H** otspot

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 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, Dar **k** (blue/gray), Ligh **t** (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, as described in the section, "Main Window Interaction".

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 throught 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 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 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. The full functionality is described in the section, "Main Window Interaction".

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, ie. 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

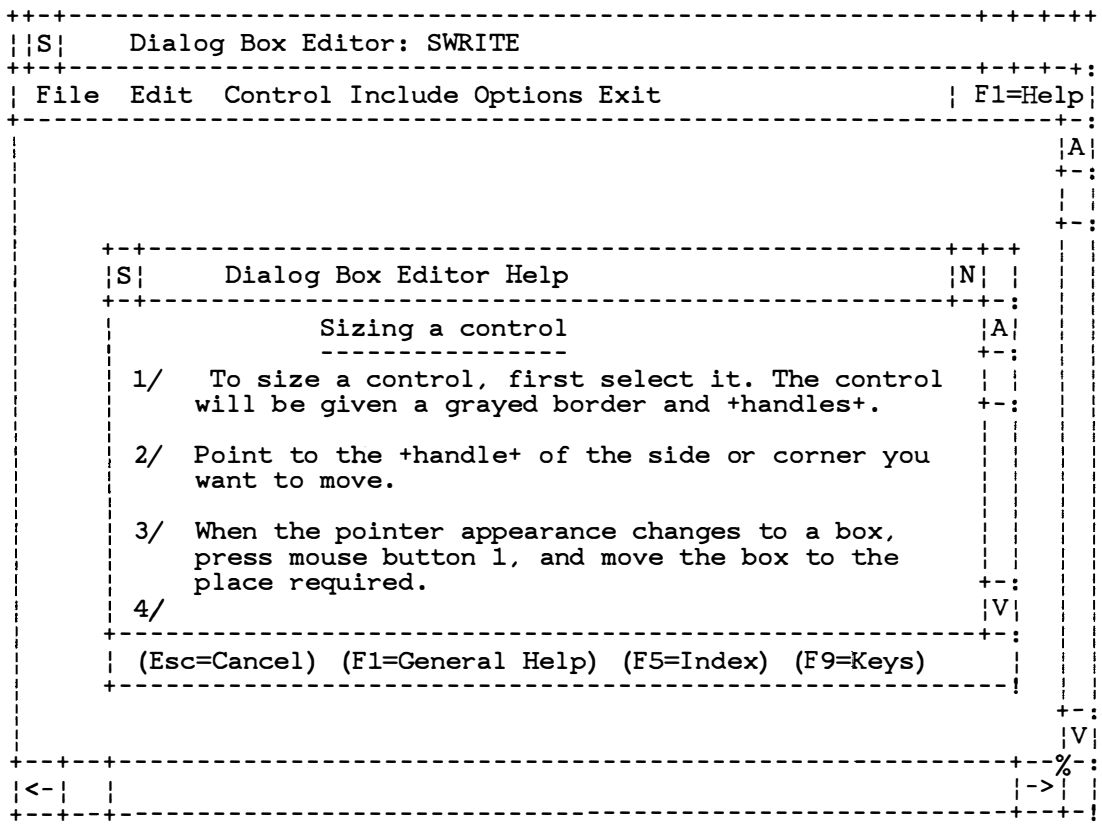
Context sensitive Help will be provided for the Icon Editor, as defined in the document Help Facility For The Dialog, Font, And Icon Editors.

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 appearence and function of the Help Facility for the Dialog, Font and Icon Editors is tha 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_CURSOR       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

- CURSOR
- ICON
- BITMAP
- FONT
- DLGINCLUDE

User-defined resources

Multiple-line statements

- STRINGTABLE
- ACCELTABLE
- MENU
- DIALOGTEMPLATE
- 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 cursors, 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
CURSOR	A cursor resource is a bitmap defining the shape of the mouse cursor 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 CURSOR, ICON, and FONT resources. The default for BITMAP resources is MOVEABLE.

filename is an ASCII string specifying the DOS filename of the file containing the resource. A full pathname must be given if the file is not in the current working directory.

Examples:

```
CURSOR pointer point.cur
CURSOR pointer DISCARDABLE point.cur
CURSOR 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 DOS filename of the file containing the cursor 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 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]
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 Presentation Manager error messages and need not be used by applications.

15.6.1.4 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>
BEGIN
  <keyval>, <cmd>, <acceloption , acceloption >
  ...
END
```

---

id — is the resource id.

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 the se 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, eg. 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.

15.6.1.5 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>
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

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
```

15.6.1.5.1 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. To insert a double quote character (") in the text, use two double quote characters ("").

The string can also contain tilde characters 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 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.

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
```

15.6.1.5.2 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
```

15.6.1.5.3 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
```

15.6.1.6 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
(BEGIN ¦ {)
     Single DIALOG, CONTROL, or WINDOW statement
(END ¦ })
```

The parts of the DIALOGTEMPLATE 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.

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 cursor 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. Hex or decimal word constants follow the CTLDATA statement, separated with commas.

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 the same as the CTLDATA statement.

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"

DIALOGTEMPLATE 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 WinLoadDialog 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
```

15.6.1.6.1 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
        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 created by a frame window. See Window Frame Architecture.

Note that the WINDOW statements in the example above could also have been a CONTROL or DIALOG statements; they are interchangable syntactically.

15.6.1.6.2 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
SS_LEFT, 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
SS_RIGHT, WS_GROUP

CTEXT

Description
Centered text control. A simple rectangle displaying the given text centered 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
SS_CENTER, 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 state- ment 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 state- ments.

Class List box

Default Style
LBS_NOTIFY, LBS_SORT, WS_VSCROLL, WS_BORDER

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 user can enter and edit text. The control displays a cursor 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_EDIT
: 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 caret. The mouse can be used to move the cursor and select characters to be replaced, or position the cursor 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_EDIT Class

Style	Meaning
ES_LEFT	Left-justified text.

ES_CENTER
Centered text.

ES_RIGHT
Right-justified text.

ES_NOHIDESEL
Normally, an edit control hides the selection when it loses the input focus, and inverts the selection when it receives the input focus. Specifying ES_NOHIDESEL overrides this default action.

WC_STATIC Class

Style Meaning

SS_LEFT
Same as LTEXT control

SS_CENTER
Same as CTEXT control

SS_RIGHT
Same as RTEXT control

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

SS_USER
User-defined item

WC_LISTBOX Class

Style Meaning

LBS_NOTIFY
: The parent receives an input message whenever the user clicks or double clicks a string.

LBS_MULTIPLESEL
: The string selection is toggled each time the user clicks or double clicks on the string. Any number of strings can be selected.

LBS_SORT
: The strings in the list box are sorted alphabetically.

LBS_NOREDRAW
: 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 cursor 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 DOS 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.

15.7 Sample programs

The sample programs give examples of how to use the Presentation Manager API.

Sample programs conform to user interface and use Presentation Manager User Controls wherever appropriate. Each sample program has a Presentation Manager window with a title bar and a menu bar containing at least an 'Exit' button which terminates the sample program.

The sample programs which have a scroll bar should use bit operations where appropriate to speed up scrolling. They should cope with devices which do not support bit operations, and also when the bit operation fails since the application is not on top.

Sample programs process redraw requests by redrawing the contents of the window.

Sample programs are written in a clear programming style with good, English comments. They are described in detail in the Presentation Manager Programming Guide or equivalent publication.

The user must be able to inspect the source of a sample program while the program is running, provided that a suitable editor is available.

The sample programs are as follows:

1. Minimal Application

   Create a window and display the message "Hello world!" using CSC functions and without using a WinProc. (Loosely based on trimmed down HELLO Microsoft Windows sample program).

   Scroll bars and menu buttons:

   - 'Exit' button

2. Typing Application

   Create a window and echo keyboard input in the window using non-retained graphics character strings. (Based on TYPE Microsoft Windows sample program).

   Scroll bars and menu buttons:

   - 'Help' button
   - 'Exit' button

3. user interface windowing.

   Demonstrate the use of each User Interface Control. The user can see and interact with each of the following: a user interface window, pull-down menus, a list box, a check box, push buttons, radio buttons, static controls, an edit control.

   Also demonstrate the use of dialogues (e.g. the open file dialogue). This may require a separate sample program to avoid making the first user interface windowing sample program too large. *Note:* The details of these sample programs should be specified by a usability expert.

   Scroll bars and menu buttons:

   - 'Help' button
   - 'Exit' button
   - Other buttons as required to exercise the User Controls.

4. Alphanumerics

   Echo keyboard input in a window using an Advanced Vio presentation space associated with the window.

   Also allow a print file to be generated containing only character information.

   Echo keyboard input as follows:

- Any alphanumerics or special character key is echoed on the screen at the cursor position which is then moved to the next character position.
- The cursor move keys and keys such as tab, backtab, newline, erase line, erase to end of line, delete line, add line, insert mode, and delete perform the corresponding operations.

Scroll bars and menu buttons:

- Vertical scroll bar
- 'Colour' button: display a pull-down to allow selection of new foreground and background colours
- 'Clear' button: clear the Advanced Vio presentation space and the window
- 'Print' button: generate a print file
- 'Exit' button

5. Non-retained graphics

   Draw a simple picture using non-retained primitives in a graphics presentation space associated with the window. Use screen coordinates, first querying the screen size to ensure that the picture is scaled and positioned suitably. However, the picture should be too tall to fit in the window vertically.

   Scroll bars and menu buttons:

   - Vertical scroll bar
   - 'Print' button: generate a print file by creating a segment, passing the picture again, issuing a spool request, and deleting the segment.
   - 'Exit' button

6. Retained graphics

   Create a simple picture consisting of at least three segments in a graphics presentation space associated with the window. The primitives in the segments include examples of each type of graphics primitive order together with a selection of attribute orders. At least one segment should call another segment.

   After creating the picture, draw it in the window. The picture should be too tall to fit in the window vertically.

   If a zoom is requested, adjust the default viewing transform and redraw the picture.

   If mouse button 1 is depressed, perform a correlate-last operation. If there is a hit on a primitive in a segment, then highlight the segment and allow the user to drag it until mouse button 1 is released. Then de-hilight the segment and place it in the picture at the new

position.

Scroll bars and menu buttons:

- Vertical scroll bar: scroll by modifying the viewing transform and redrawing the picture
- 'Print' button: generate a print file
- 'Exit' button

7. Fonts

   Create a window and display a horizontal line of alphanumeric and special characters in the window using a font. (Based on FONTTEST Microsoft Windows sample program).

   Scroll bars and menu buttons:

   - 'Commands': pull-down containing the options:
     - 'Sample': draw the line of characters again below any other lines of characters
     - 'Clear': clear the window
     - 'Options': allow the user to choose new font attributes using a dialogue box. Characters are subsequently drawn with these attributes.
   - 'Help' button
   - 'Exit' button

8. Bitmaps and image

   Draw non-retained graphics into a bitmap, retrieve the bitmap data into the application, and then draw this image data in the window.

   Draw non-retained graphics into a bitmap and then bitblt this into a window.

   Scroll bars and menu buttons:

   - 'Exit' button

9. Clipboard

   Create a window and display text in it using the gpi. Allow the user to interchange data between instances of this application using a cut/copy/paste editing metaphor. The user should select areas of text, and the application will place it into the clipboard in the CF_DSPTEXT and CF_TEXT formats. The CF_TEXT format will be delay rendered.

   When the user selects paste, the application will copy the CF_TEXT format from the clipboard.

   Scroll bars and menu buttons:

- 'Help' button
- 'Exit' button
- 'Select' button: allow the user to select a rectangle of characters VIO data using the mouse. Show which rectangle is selected by displaying it in reverse video.
- 'Copy', 'Cut', 'Paste', and 'Clear' buttons: allow the user to do the corresponding clipboard operations on the selected data.

10. Template application

    This is a program which serves as a template for application authors. The parts of the program which should be replaced by application code are delimited by suitable comments. (Based on TEMPLATE Microsoft Windows sample program).

    Scroll bars and menu buttons:

    - 'Help' button
    - 'Exit' button

Chapter 16
Device Drivers

16.1 Device Driver Interface

16.1.1 Overview

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.

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.

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

16.1.3.1 Stack Arguments

Four arguments are passed on the stack to the major function handlers. These arguments are the same for all handlers. Each argument is a 32 bit quantity. They are pushed onto the stack in the order: lpArgs, Command, hDC, FunN. As an example, the OutputLine major function handler would be called as:

16.1.3.1.1 OutputLine(lpArgs, Command, hDC, FunN)

Parameters;

lpArgs — A far pointer to the original arguments to the graphics engine. The device driver gets the original arguments, as defined in the graphics engine specification, but in a structure rather than on the stack.

The selector part of lpArgs is always the device driver's stack segment. This means that the device driver can potentially avoid a segment selector load when accessing the arguments by using SS directly.

Command — Bit flags indicating what operations need to be performed.

Bit 0 => COM_DRAW
Draw the figure.

Bit 1 => COM_BOUND
Calculate the bounding rectangle.

Bit 2 => COM_CORR
Calculate correlation with the Pick Window.

(These three functions only apply to drawing

functions and should be ignored for other functions.)

Bit 3 => COM_BITMAP
Set by the device driver prior to a call back to the engine using SimulationEntry to ensure that the function gets routed to the bitmap simulation routines.

Bits 16 - 23
may be defined by a simulation for its own use.

Bits 24 - 31
may be defined by the driver for its own use.

All other bits should be ignored.

hDC This is a pointer to the actual DC in memory. The high word is the DC Segment (for all DC's), the low word is the offset of this DC.

The device driver is given access to a DWORD (4 bytes) of information starting at byte offset 4 in the DC. This DWORD 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 DWORD 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.

FunN The function number. The minor function number is in the low word. The major function number is in the low byte of the high word. The high byte should be ignored.

16.1.3.2 Command Bits for Drawing, Correlation and Bounds Calculation

Three command bits are passed to the device driver and may be used in any combination. These bits are:

- COM_DRAW
- COM_BOUND
- COM_CORR

requesting that the device perform drawing, bounding, and correlating respectively.

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.

16.1.3.2.1 Draw

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).

16.1.3.2.2 Bound

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, through SimulationEntry.

All device drivers must be able to calculate bounds on any figure they can draw.

16.1.3.2.3 Correlation

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 error code ERROR_CORRELATION_HIT (07FF) in AX.

Only display device drivers are required to calculate correlations.

16.1.3.3 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.

16.1.3.4 Transforms

Primitive functions (eg PolyLine) will be passed to the driver in world co-ordinates. The driver may either perform the transformations itself, or it may use Convert (via SimulationEntry), to get the engine to do them for it.

Some functions which are generated internally (eg ScanLR), will, however, already be in device co-ordinates when passed across the DDI. The driver can tell whether any particular function has already been transformed or not by inspecting a bit in the command flags. (This bit is to be defined.)

All device co-ordinates (ie after transformation) passed across the DDI are 32 bit fixed point numbers, with 16 bits integer and 16 bits fraction. Any co-ordinates in world, model, page, or default page space are 32 bit signed integers.

16.1.3.5 Return Values

The device driver must return an error code in the AX register when it has completed the function call. A return value of zero indicates successful completion. Any other value indicates that an error or other noteworthy condition has occurred. Defined return codes are:

0000	OK completion.
07FF	Primitive passed through pick window.
08FF	Two correlation hits on EndArea primitive.
2400	Set Model Transform matrix element overflow.
3100	Set Viewing Transform matrix overflow.
EE00	End of data. (Used by some enumerating functions.)

16.1.3.6 Register Content Preservation

AX will contain any error code. Registers BX, CX, DX, and ES may be destroyed. All other registers must be preserved.

16.1.3.7 Calling SimulationEntry

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:

```
SimulationEntry( lpArgs, Command, hDC, FunN )
```

The arguments are exactly what would be passed to a driver or simulation.

The Command word may be set to request any combination of COM_DRAW, COM_CORR, and COM_BOUND. In addition, a driver can set bits of its own definition. What this means is that all simulations will pass bits 24 - 31 of the Command unchanged to any calls they make. In this way, a driver can make a call to a simulation with some of its own Command bits set, and recognize any driver calls that are generated by that simulation.

In the same way, drivers must pass bits 16 - 23 of the command that they received unchanged to any simulations that they call. This way a simulation can recognize when a driver is calling it from inside another simulation call.

16.1.3.8 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.

16.1.3.9 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, as follows:

170001 CreateJournal (lphJournal,FunN)
Creates a journal. A handle is returned.

170002 DeleteJournal (hJournal,FunN)
Deletes a journal.

170003 StartJournal (hDC,hJournal,FunN)
Tells the engine that all drawing and attribute calls for the given DC are to be recorded in the journal. After recording a drawing call in the journal, the engine will pass the

drawing call on to the device driver, but without the COM_DRAW command bit.

170004 StopJournal (hDC,hJournal,FunN)
Tells the engine that drawing and attribute calls should no longer be recorded in the journal.

170005 PlayJournal (Control,hDC,hJournal,FunN)
Tells the engine to play back the journal to the given DC. The Control argument determines how many calls to play back. This routine returns an error code when the end of the file is reached. The next call after the end of file is reached will start the file again from the beginning.

16.1.3.10 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 DOS 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 Presentation Manager API.

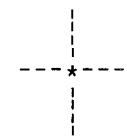
16.1.3.11 Cursors

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 */
{
   WORD      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
{
   WORD      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();
{
   WORD      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.

16.1.4 Dispatching Minor Functions

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.

Note: these numbers are subject to change.

16.1.4.1 OutputArc 00

0300 0000	GetArcParameters
0300 0001	SetArcParameters
0300 0002	Arc
0400 0003	FullArcInterior
0400 0004	FullArcBoundary
0400 0005	FullArcBoth
0600 0006	PartialArc
0600 0007	ArcDDA
0600 0008	FilletDDA
0600 0009	PartialArcDDA
0400 000A	PolyFillet
0600 000B	PieSliceInterior
0600 000C	PieSliceBoundary
0600 000D	PieSliceBoth
0300 000E	BoxInterior
0300 000F	BoxBoundary
0300 0010	BoxBoth
0300 0011	QueryArcDDA
0300 0012	QueryFilletDDA
0300 0013	QueryPartialArcDDA
0400 0014	PolySpline
0400 0015	PolyFilletSharp

16.1.4.2 OutputLine 01

0401 0000	PolyLine
0301 0001	PolyShortLine
0601 0002	LineDDA
0301 0003	GetCurrentPosition
0301 0004	SetCurrentPosition

0301 0005	QueryLineDDA

16.1.4.3 OutputMarker 02

0402 0000	PolyMarker

16.1.4.4 OutputScan 03

0303 0000	ScanLR
0503 0002	PolyScanLine

16.1.4.5 OutputFill 04

0304 0000	FloodFill
0404 0002	AltPolygon
0404 0003	WindPolygon

16.1.4.6 Bitmap 05

0505 0000	DeviceCreateBitmap
0405 0001	DeviceDeleteBitmap
0305 0002	DeviceSelectBitmap
0305 0003	GetBitmapParameters
0705 0004	GetBitmapBits
0705 0005	SetBitmapBits
0305 0006	GetPel
0305 0007	SetPel
0505 0008	ImageData
0705 0009	Bitblt
0505 000A	DeviceSetCursor
0405 000B	SaveBits

16.1.4.7 Textout 09

0709 0000	CharStringCtrl
0509 0000	CharString
0409 0002	CharRect
0409 0003	CharStr
0409 0004	ScrollRect
0309 0005	UpdateCursor
0509 0006	QueryTextBox
0709 0007	QueryTextBreak

16.1.4.8 Area 0A

030A 0000	BeginArea
030A 0001	EndArea
000A 0002	AccumulateArea
040A 0003	BeginClipArea
030A 0004	EndClipArea
030A 0005	BeginStrokes
030A 0006	EndStrokes
020A 0007	QueryAreaState
050A 0008	DrawFrame

16.1.4.9 Bounds 0B

030B 0004	QueryCharCorr
030B 0005	GetPickWindow
030B 0006	SetPickWindow

16.1.4.10 Clip 0C

040C 0000	GetClipBox
050C 0001	SelectClipRegion
040C 0002	IntersectClipRectangle
040C 0003	ExcludeClipRectangle
030C 0004	OffsetClipRegion
030C 0005	SetXformRect
030C 0006	QueryClipRegion
040C 0007	PtVisible
040C 0008	RectVisible
050C 0009	GetClipRects
050C 000A	SelectVisRegion
030C 000B	QueryVisRegion

16.1.4.11 Region 0D

050D 0000	GetRegionBox
060D 0001	GetRegionRects
050D 0002	CreateRectRegion
030D 0003	DestroyRegion
050D 0004	SetRectRegion
070D 0005	CombineRegion
040D 0006	OffsetRegion
050D 0007	EqualRegion
050D 0008	PtInRegion
050D 0009	RectInRegion
030D 000A	PaintRegion

16.1.4.12 Transform 0E

060E 0000	Convert
040E 0001	GetModelXform
040E 0002	SetModelXform
040E 0003	GetWindowViewportXform
040E 0004	SetWindowViewportXform
030E 0005	GetGlobalViewingXform
040E 0007	SetGlobalViewingXform
030E 0008	GetGraphicsField
030E 0009	SetGraphicsField
030E 000A	GetPageUnits
050E 000B	SetPageUnits
030E 000C	GetPageWindow
040E 000D	SetPageWindow
030E 000E	GetPageViewport
040E 000F	SetPageViewport
000E 0010	GetDCOrigin
030E 0012	SetDCOrigin
030E 0013	GetViewingLimits
030E 0014	SetViewingLimits

16.1.4.13 Attributes 0F

030F 0001	EnableKerning
040F 0002	GetKerningPairTable
040F 0003	GetTrackKernTable
060F 0004	SetKernTrack
060F 0005	DeviceSetAttributes
040F 0006	DeviceSetGlobalAttribute
040F 0007	NotifyClipChange
070F 0008	RealizeFont

020F 0009	ErasePS
030F 000B	GetDCCaps
040F 000C	DeviceQueryFontAttributes
040F 000D	DeviceQueryFonts
070F 000E	DeviceQueryFontSpace
020F 000F	GetPatternOrigin
030F 0010	SetPatternOrigin
030F 0011	SetStyleRatio
050F 0012	SetLineTypeGeom
050F 0013	QueryLineTypeGeom

16.1.4.14 Color 10

0410 0000	QueryColorData
0610 0001	QueryLogColorTable
0710 0002	CreateLogColorTable
0210 0003	RealizeColorTable
0210 0004	UnrealizeColorTable
0610 0005	QueryRealColors
0510 0006	QueryNearestColor
0510 0007	QueryColorIndex
0510 0008	QueryRGBColor

16.1.4.15 Query 11

0411 0000	QueryDeviceBitmaps
0411 0001	QueryDeviceCaps
0711 0003	Escape
0511 0005	QueryHardcopyCaps

16.1.4.16 Device Mode 14

0614 0005	DeviceMode

The device driver is allowed to reject any call it does not want to handle, as long as it is not one of the "required" functions.

The device driver must pass any call it does not handle to the default simulation for the major handler being called.

16.1.5 The Dispatch Table

16.1.5.1 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 lDeviceDispatchTable. This table will already be filled with the addresses of the graphics engine default major handlers. (Copied from the DefaultDispatchTable.) The device driver must overwrite the entries in this table that correspond to the functions it wishes to handle.

The device driver must handle all functions listed in the table of required functions. The remaining functions are optional.

16.1.5.2 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
- SetAttributes
- SetGlobalAttributes

16.1.5.3 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.

16.1.6 Primary Function Definitions

This section contains the definitions of the device driver functions which must be called via the normal dynamic link mechanism rather than through the Function Handler mechanism. These entry points must be exported by the device driver. These are required for all device drivers.

16.1.6.1 Enable

The Enable function performs initialization of the device driver, the physical device, and device contexts. It is called as:

Enable(U32_ SUBFUNCTION , P32_ PARAMS , P32_ RETURNS)

16.1.6.1.1 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 relevent 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.

16.1.6.1.2 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 (eg "EPSON")

p32_devicename
Pointer to ASCIIZ name of the device (eg "<TBD>")

p32_outputname
Pointer to ASCIIZ name of the output device. This may be either the spooler output class (eg "PRINT" or "PLOT"), or the device name for the physical port (eg "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 "MX-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.

16.1.6.1.3 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 (eg "EPSON")

  p32_devicename
  : Pointer to ASCIIZ name of the device (eg "<TBD>")

p32_outputname
: Pointer to ASCIIZ name of the output device. This may be either the spooler output class (eg "PRINT" or "PLOT"), or the device name for the physical port (eg "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 "MX-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.

16.1.6.1.4 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 "MX-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.

16.1.6.1.5 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.

16.1.6.1.6 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.

16.1.6.1.7 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.

16.1.6.1.8 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.

16.1.6.1.9 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.

16.1.6.1.10 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.

16.1.6.1.11 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 Managere 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.

16.1.6.1.12 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 hanlder's address in case it can't handle the function completely.

16.1.7 Definitions of Functions called via Handlers

A major function handler of the device driver is called with an lpArgs parameter which points to a structure containing the arguments for the minor function. This structure is defined for each minor function below.

16.1.7.1 OutputArc Functions - Major Function 00.

Function: 0300 0000 GetArcParameters

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_AttributeData
```

```
};
```

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.

Function: 0300 0001 SetArcParameters

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_AttributeData
};
```

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 half the major 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 Co-ordinates. Any other non-square transformations in force will change the shape of the figure accordingly.

Function: 0300 0002 Arc

```
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
```

Upon completion, the current x,y position is the third position of the arc.

Function: 0400 0003 FullArcInterior

See below.

Function: 0400 0004 FullArcBoundary

See below.

Function: 0400 0005 FullArcBoth

```
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 anticlockwise 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.

The current x,y position is not changed by FullArc.

Function: 0600 0006 PartialArc

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  s32_te
    ULONG  s32_ts
    ULONG  s32_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 GpiFullArc. 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 centre, 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 co-ordinate pair which are the co-ordinates 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.

Upon completion, the current x,y position is set to the final point of the arc.

Function: 0600 0007 ArcDDA

See Below.

Function: 0600 0008 FilletDDA

See below.

Function: 0600 0009 PartialArcDDA

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Continue
    ULONG* p32_Pels
    ULONG* p32_PelCount
    ULONG* p32_WorkArea
};
```

Computes points along a primitive drawn through the points passed, under the current transforms, and returns the x,y coordinate values in device coordinates.

Points are computed and returned in batches, according to the p32_PelCount parameter. It can be assumed that the points defining the primitive do not change during a sequence.

The data in the work area begins with four zero bytes (which the engine may update as the DDA progresses), is followed by the points defining the primitive the DDA is being applied to, depending on the DDA function being requested, then by more reserved bytes, up to the length returned by Query DDA.

The points defining the primitive are as follows:

Line — Two x,y coordinate pairs are passed

Fillet — Three x,y coordinate pairs are passed

Arc — Three x,y coordinate pairs are passed

Partial Arc
: Parameters defining the partial arc centre, multiplier, start and end angles as follows:

```
s32_startX      X Coordinate of start of line
s32_startY      Y Coordinate of start of line
s32_centreX     X Coordinate of centre of partial arc
```

```
s32_centreY      Y Coordinate of centre of partial a
s32_multiplier   To apply to arc parameters p,q,r,s
s32_startAngle   Angle of start of arc
s32_endAngle     End angle.  Start=End means full ar
```

Parameters:

p32_WorkArea
: Pointer to work area for this function, starting four zero bytes, then the parameters listed above, then more work area.

p32_PelCount
: Long pointer to a u32 count of the pels to return. At least this many points must be present in the p32_Pels array. Fewer points will be returned at the end of a DDA. In this case, the count will be updated to show the number of pels actually returned.

p32_Pels
: Pointer to array of x,y coordinates to return the next DDA points into.

p32_Continue
: Pointer to a u32 variable which will be set to 0 if the DDA is complete, otherwise 1.

The current x,y position is not affected by DDA.

Function: 0400 000A PolyFillet

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  s32_n
    ULONG* p32_xy
};
```

Creates a poly primitive, which can be a fillet, sharp 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.

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.

Upon completion, the current x,y position is the last point in the array of x,y coordinates.

Function: 0300 000E BoxInterior

See below

Function: 0300 000F BoxBoundary

See below

Function: 0300 0010 BoxBoth

```
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, or neither. 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.

Function: 0300 0011 QueryArcDDA

See below

Function: 0300 0012 QueryFilletDDA

See below

Function: 0300 0013 QueryPartialArcDDA

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_bytes
};
```

This returns the number of bytes of work area required by a particular DDA function (line, partial arc, fillet, arc). The application must allocate this much work area to be able to use the DDA function.

Different function numbers are used to query arc, fillet, line and partial arc DDAs.

Parameters:

p32_bytes
: Long pointer to the u32 variable which will hold the return value.

Function: 0400 0014 PolySpline

See section on PolyFillet for details.

Function: 0400 0015 PolyFilletSharp

```
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 x,y pairs identify the end points of the lines.

This function is similar to PolyFillet, except that instead of allowing the implementation to choose the sharpness of each of the constituent fillets, these are specified explicitly.

The sharpness of each fillet is defined as follows. Let A and C be the start and end points, respectively, of the fillets, 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 WD/DB.

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/3) elements, each array element being a s32_sharpness parameter. Each value, when divided by 65536, gives the sharpness of successive fillets.

```
>1.0 means a hyperbola is drawn
=1.0 means a parabola is drawn
<1.0 means an ellipse is drawn
```

Upon completion, the current x,y position is the last point in the array of x,y coordinates.

16.1.7.2 OutputLine Functions - Major Function 01.

16.1.7.2.1 Function: 0401 0000 PolyLine

See section on PolyFillet for details.

16.1.7.2.2 Function: 0301 0001 PolyShortLine*

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_PolyShortData
};
```

Draws a set of lines encoded as a series of steps. This call is not revealed in the API.

Parameters:

---

p32_PolyShortData

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.

16.1.7.2.3 Function: 0601 0002 LineDDA

16.1.7.2.4 Function: 0301 0003 GetCurrentPosition

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_AttributeData
};
```

Returns the current position as an x,y coordinate pair:

(s32_x, s32_y)

Parameters:

p32_AttributeData
: Specifies the return address for the data.

16.1.7.2.5 Function: 0301 0004 SetCurrentPosition

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_xy
};
```

Sets the current x,y position to the specified value. This has the following side-effects:

- The line type sequence is reset.
- If the current context is 'in area', a figure closure line is generated if necessary. This may generate a correlation hit to occur. if necessary. this may cause a correlation hit on an area bound ary

The current position is correlated on and/or merged into the bounds, if it is actually used in a drawing primitive. So, for example, the sequence SetCurrentPosition to P1, SetCurrentPosition to P2, Polyline to P3, will not merge P1 into the bounds or correlate on it, but will merge P2 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.

16.1.7.2.6 Function: 0301 0005 QueryLineDDA

16.1.7.3 OutputMarker Functions - Major Function 02.

16.1.7.3.1 Function: 0402 0000 PolyMarker

16.1.7.4 OutputScan Functions - Major Function 03.

16.1.7.4.1 Function: 0303 0000 ScanLR*

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_SearchData
};
```

Scans left or right from a given location looking for a pel which satisfies a search condition.

Parameters:

p32_SearchData
Pointer to the search data structure, defined as follows:

s32_Startx
Start x coordinate for search

s32_Starty
Start y coordinate for search

s32_HitX
Found point x coordinate

s32_HitY
Found point y coordinate

u32_Color
Index of color of pel for search

s32_Control
Control flags:

— D0 = 0 ; Search for not Color

— D0 = 1 ; Search for Color

— D1 = 0 ; Step right

— D1 = 1 ; Step left

16.1.7.4.2 Function: 0503 0002 PolyScanLine*

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_BoundingRect
    ULONG* p32_PSL2
    ULONG* p32_PSL1
};
```

Fills an area lying between two polyshortlines.

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

Whenver direction = 3 or 7 (ie 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.

This function must be supported by all device drivers.

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.

16.1.7.5 OutputFill Functions - Major Function 04.

16.1.7.5.1 Function: 0304 0000 FloodFill

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Color
};
```

This fills an area of the device with the current pattern attributes. The area starts at the current position, and extends in all directions until it comes to pels of the specified colour.

Note that the results produced by this function are highly device dependent.

Parameters:

---

u32_Color
: Specifies the color index for the color which bounds the filled area.

16.1.7.5.2 Function: 0404 0002 AltPolygon

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  s32_n
    ULONG* p32xy
};
```

Draws a filled polygon. The boundary is drawn with the current line attributes. The interior is filled with the current fill attributes in Alternating mode - i.e. every other enclosed region in a complex polygon is filled.

Parameters:

p32_xy
: Points to an array of x,y point coordinates.

  An extra x,y pair is passed at the start of the array and is not included in the number of pairs for working space. The starting (and ending) point is listed only once. The whole array can be overwritten by transformed coordinates, if necessary.

s32_n
: Specifies the number of x,y pairs.

The current position is unchanged by this call.

16.1.7.5.3 Function: 0404 0003 AltPolygon

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  s32_n
    ULONG* p32xy
};
```

Draws a filled polygon. The boundary is drawn with the current line attributes. The interior is filled with the current fill attributes in Winding mode - i.e. only 'enclosed' regions in a complex polygon are filled. 'Enclosed' is defined by the boundary of the region being traversed clockwise one or time than it is traversed anticlockwise (or vice-versa) - i.e. a non-zero 'winding number', which is incremented for each clockwise traversal and decremented for each anticlockwise traversal.

Parameters:

p32_xy Points to an array of x,y point coordinates.

An extra x,y pair is passed at the start of the array and is not included in the number of pairs for working space. The starting (and ending) point is listed only once. The whole array can be overwritten by transformed coordinates, if necessary.

s32_n Specifies the number of x,y pairs.

The current position is unchanged by this call.

16.1.7.6 Bitmap Functions - Major Function 05.

16.1.7.6.1 Function: 0505 0000 DeviceCreateBitmap*

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Parms
    ULONG  u32_Usage
    ULONG* p32_Handle
};
```

Asks the device driver to create a bitmap. The device driver may create it in system RAM, or in its own hardware. The bitmap may be kept in any format. The device driver returns a handle for the bitmap. The handle is private to the device and will be used only to identify the bitmap in the calls DeviceDeleteBitmap and DeviceSelectBitmap.

Optionally, the device may be requested to initialize the bitmap. See below.

Return codes:

```
AX = Error_No_Memory
```

Parameters:

p32_Parms
Points to a list of parameters for the bitmap. The parameter list contains the following:

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
: is a set of bit flags controlling the function:

— Bit 0 => Keep a memory backup while in device storage.

— Bit 1 => This bitmap may not be discarded.

— Bit 2 => Initialize the bitmap by translating a standard format bitmap. If this bit is set, p32_Handle points to a pair of selector values. The selector at p32_Handle[0], combined with an offset of 0000 points to a standard format bitmap data. The selector at p32_Handle[2], combined with the offset 0000 points to the parameters of the standard bitmap. These are: height, width, planes, and bits per pixel. Following these parameters is a color table defining the RGB color value of the color indices stored for each pixel. However, if (planes)*(bits per pixel) is already 24 bits, then no color table is present. In this case, an RGB value is already stored for each pixel. Both selectors must be freed by this call.

p32_Handle
: Points to a location where the device's handle for the resulting bitmap is to be stored. If bit 2 of Usage is set, then it initiallly points to two selectors to be used to initialize the bitmap. After the data is used, the selectors must be freed, and then overwritten with the device's bitmap handle.

16.1.7.6.2 Function: 0405 0001 DeviceDeleteBitmap*

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Usage
    ULONG* p32_Handle
};
```

Asks the device driver to delete a bitmap that it created. The given handle is the one returned from DeviceCreateBitmap.

Optionally, the device may be asked to translate the bitmap to a standard format before deleting it.

Return codes:

```
AX = Error_No_Memory
AX = Error_Bad_Handle
```

Parameters:

Usage Is a set of bit flags controlling the function:

— Bit 2 => Before deleting the bitmap, translate it into one of the standard formats. Two blocks of memory must be allocated to return the data, one for the bitmap data, and the other for the bitmap parameters and color translation table. The device driver is free to choose which standard format to translate into, but must take into account the parameters the application originally requested in the DeviceCreateBitmap call. In general, the device should try to use the format that requires the smallest space to store and does not lose any information presently in the bitmap.

p32_Handle
points to a location where the device's handle for the bitmap is stored. If bit 2 of Usage is set, then the call should store the selector for the data area of the translated bitmap in lphandle[0], and the selector for the parameter area in lpHandle[2].

16.1.7.6.3 Function: 0305 0002 DeviceSelectBitmap*

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Handle
};
```

Tells the device driver that a new bitmap is being selected into the given DC. Unlike the engine version of this call, it does not need to return the previously selected bitmap, since the engine already knows it.

Return codes:

```
AX = ERROR_BAD_HANDLE
```

Parameters:

Handle Is the bitmap handle that the device driver returned when the bitmap was created.

16.1.7.6.4 Function: 0305 0003 GetBitmapParameters

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Parm
};
```

This returns various parameters, associated with performing bitmap operations on the specified Device Context, which must either be a memory Device Context with a selected bitmap, or a Device Context for a device which supports raster operations.

Parameters:

p32_Parm
Provides a long pointer to a data area in which the returned parameters are placed. The returned parameters are:

u32_width, u32_height
Return the width and height of the bitmap in pels respectively.

u32_Planes
Returns the number of colour planes in the bitmap.

u32_Bitcount
Returns the number of adjacent colour bits per pel in the bitmap.

16.1.7.6.5 Function: 0705 0004 GetBitmapBits

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Info
    ULONG* p32_Address
```

```
    ULONG  u32_BitCount
    ULONG  u32_Planes
    ULONG  u32_Length
    ULONG  u32_Offset
};
```

This function 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 format of the data is as indicated by the data returned by the GetBitmapParameters call.

If any of the source data is not available, for example if the Device Context is connected to a screen window which is not currently all visible, no error code is returned and the operation proceeds.

There are several standard bitmap formats which the data is likely to be in.

Parameters:

---

u32_Offset
: Specifies the 4-byte offset in bytes into the bitmap data from which the transfer must start. This is used when the bitmap data is too long to fit into a single application buffer.

u32_Length
: Specifies the 4-byte length in bytes of the bitmap data to copy.

u32_Planes
: The number of color planes in the bitmap. For reference: each plane has ((Width*Bitcount+31)/32*4*Height) bytes. If this parameter is zero then data will be in internal device format and the bitcount parameter will be ignored.

u32_Bitcount
: The number of adjacent color bits per pel. The Planes and Bitcount parameters indicate the format of the data that is to be returned.

p32_Address
: specifies the address in application storage into which the bitmap data is copied.

p32_Info
: A pointer to the Info table (see below)

Standard Bitmap Formats

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 lowest order 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 {
  UINT  BitmapWidth;       /* length of a scanline                        */
  UINT  BitmapHeight;      /* number of scanlines                         */
  UINT  BitmapPlanes;      /* number of planes (1 if standard format)     */
  UINT  BitmapBitcount;    /* number of bits per pixel                    */
  RGB   BitmapColors[];    /* color table                                 */
};
```

The BitmapColors array is a packed array of 24 bit RGB values. If there are N bits per pixel, then the BitmapColors 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 BitmapColors 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   =
'23'X '12'X  '30'X '00'X          /* bottom line */
'31'X '23'X  '10'X '00'X          /* middle line */
'12'X '31'X  '20'X '00'X          /* 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
};
```

16.1.7.6.6 Function: 7005 0005 SetBitmapBits

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Info
    ULONG* p32_Address
    ULONG  u32_BitCount
    ULONG  u32_Planes
    ULONG  u32_Length
    ULONG  u32_Offset
};
```

This function transfers bitmap data from application storage into the specified Device Context, which must be a memory Device Context with a selected bitmap.

Parameters:

---

u32_Offset
: Specifies the 4-byte offset in bytes into the bitmap data from which the transfer must start. This is used when the bitmap data is too long to fit into a single application buffer.

u32_Length
: Specifies the 4-byte length in bytes of the bitmap data to copy.

u32_Planes
: The number of color planes in the bitmap. For reference: each plane has ((Width*Bitcount+31)/32*4*Height) bytes. If this parameter is zero then data will be in internal device format and the bitcount parameter will be ignored.

u32_Bitcount
: The number of adjacent color bits per pel. The Planes and Bitcount parameters indicate the format of source data.

p32_Address
: specifies the address in application storage from which the bitmap data is copied.

p32_Info
: A pointer to the Info Table (See GetBitmapBits for details)

Standard Bitmap Formats - See the GetBitmapBits section for a full descripton of Standard bitmap formats.

16.1.7.6.7 Function: 0305 0006 GetPel

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Parm
};
```

This function gets a pel from a specified position.

Parameters:

---

p32_Parm
: Provides a long pointer to a parameter block:

  u32_X, u32_Y
  : Specify x and y values for the pel position.

  u32_Color
  : Returns a color index value for the colour of the pel.

16.1.7.6.8 Function: 0305 0007 SetPel

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Parm
};
```

This function sets a pel at a specified position to the current line attribute color and mix.

Parameters:

---

p32_Parm
: Provides a long pointer to a parameter block:

u32_X, u32_Y
Specify the pel position.

16.1.7.6.9 Function: 0505 0008 ImageData

```
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 of a row are zero.

Parameters:

p32_Data
Points to a string of image data with one bit per pel. The first pel's data is in the first bit of the first byte of data.

s32_n Specifies the number of data bits to output from p32_Data.

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 device, and so on.

The current position is not affected by this order.

16.1.7.6.10 Function: 0705 0009 Bitblt

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Target
    ULONG  u32_Style
    ULONG  u32_Mix
    ULONG* p32_Parm
    ULONG  u32_Count
    ULONG  u32_Source
};
```

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) 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.

Parameters:

---

u32_Source
: Specifies the handle of the source Device Context.

u32_Count
: Specifies the number of x,y pairs of coordinates in the parameter block.

  - For Style = 0 AND Mix value specifying PatternBlt only, count must be >= 2.
  - For style = 0 (BitBlt) count must be >= 3.
  - For style 1, 2 or 3 (StretchBlt) count must be >= 4.

  Note that count > 4 is only useful if the device driver supports a non-standard use mode. However, the driver should allow for there to be more than 4 coordinate pairs, even if it does not use them.

p32_Parm
: Provides a long pointer to the parameter block:-

  ---

  u32_TargX1, u32_TargY1, u32_TargX2, u32_TargY2,
  : Specify the bottom left and top right corners of the target rectangle.

u32_SrcX1, u32_SrcY1, u32_SrcX2, u32_SrcY2
: Specify the bottom left and top right corners of the source rectangle.

  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 '00'..'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 bit-map 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)
```

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 system. Values greater than 32K are passed directly to the device driver. This allows applications to use values of their own for use with "intelligent" devices.

u32_Target
: Specifies the handle of the target Device Context.

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.

16.1.7.6.11 Function: 0505 000A DeviceSetCursor

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Command
    ULONG* p32_Args
};
```

Sets the cursor bitmap that defines the cursor shape. Each call replaces the previous bitmap with that pointed to by SC_arg_hbmCursor. If SC_arg_hbmCursor is null, the cursor has no shape and its image is removed from the display screen.

Parameters:

lpArgs points to an argument structure as follows:

```
SetCursor_arg_struc struc
    SC_arg_FunN         dd  ?
    SC_arg_hDC          dd  ?
    SC_arg_hbmCursor    dd  ?
    SC_arg_yHotspot     dd  ?
    SC_arg_xHotspot     dd  ?
SetCursor_arg_struc ends
```

SC_arg_hbmCursor
: The bitmap handle to be used for the cursor image.

SC_arg_yHotspot
: The y position within the cursor of the "hot spot."

SC_arg_xHotspot
: The x position within the cursor of the "hot spot."

Return value:

```
ax = 0 if the function executed successfully
ax = -1 otherwise
```

16.1.7.6.12 Function: 0405 000B SaveBits

```
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
```

Return value:

```
ax = 1 success
ax = 0 failure
```

16.1.7.7 Textout Functions - Major Function 09.

Note: The functions CharRect, CharStr, and ScrollRect need a mapping from the Avio symbol set indices 0..3, to the actual pixel images for these indices. This is achieved as follows:-

- When Avio symbol sets are loaded, the graphics engine will also dispatch a call to the device drivers DeviceSetAttributes call, with a bundle typeof character attributes.
- The char_set field will be set to 1-3, and the char_font field will contain a long cp segto the symbol set, translated by the engine to look like a font.

- At this point the driver will save this pointer.
- Therefore the device driver will have 4 "fonts/symbol sets" selected at once (3 for avio, and one for graphics).

16.1.7.7.1 Function: 0709 0000 CharStringPos

```
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, with user controlled spacing.

Parameters:

---

p32_xy Long pointer to the an array of four coordinates, defining the opaque rectangle. The first two coordinates of this array are spare, the second two will contain the coordinates of one corner of the rectangle. The other corner of the rectangle is at the current position. The opaque rectangle will ignore any background mix attributes, and is drawn using over-paint 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 Width vector not present
        = 1 Width vector present

Bit 2   = 0 Normal text
        = 1 Grayed (de-emphasised) 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 is not present, then bits 0 and 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. These numbers will be used instead of the real widths of the characters. There will be s32_n numbers in this array. Long pointer to an array of s32 numbers, the character increments. These numbers are used instead of the real widths of the characters. 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.

16.1.7.7.2 Function: 0509 0001 CharString

```
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.

16.1.7.7.3 Function: 0509 0002 CharRect

```
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 a 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 and

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.

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.

16.1.7.7.4 Function: 0509 0003 CharStr

```
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 a 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 and

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.

16.1.7.7.5 Function: 0409 0004 ScrollRect

```
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 and

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.

16.1.7.7.6 Function: 0309 0005 UpdateCursor

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_PS
};
```

Sets the alphanumeric cursor position, shape and visibility.

---

p32_PS points to the Vio presentation space.

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. Its 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.

16.1.7.7.7 Function: 0509 0006 QueryTextBox

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_xy
    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 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.
p32_xy	Long pointer to the return array of 5 x,y pairs.

16.1.7.7.8 Function: 0709 0007 QueryTextBreak

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_rem
    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.
p32_rem	Long pointer to a s32 variable to return the amount of space in the extent which will be unused if only p32_n characters are kept in the string.

16.1.7.8 Area Functions - Major Function 0A.

When a device driver takes over the Area function group, it must be able to handle area fill with boundary drawing. If it needs to have the boundary drawing orders repeated to achieve this, then it can place the line drawing calls within a strokes bracket, filling with each original call, and drawing the boundary with the close of the strokes bracket.

If the device is only able to handle the fill it should

1. Hook the BeginArea call.
2. If there is a border, the driver issues a BeginStrokes call back into the engine.

3. For each line, fillet, arc etc. which the driver receives it uses the primitive to figure out its fill.
4. When the EndArea order is received, the driver does its fill and also send off an EndStrokes to the engine.
5. The strokes process will handle the resulting border as a line and draw it either
    1. As a series of cosmetic primitives
    2. By generating a new area (without border) for geometric thick lines

16.1.7.8.1 Function: 030A 0000 BeginArea

```
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.

16.1.7.8.2 Function: 030A 0001 EndArea

```
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 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).

16.1.7.8.3 Function: 030A 0002 AccumulateArea

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Cancel
};
```

16.1.7.8.4 Function: 040A 0003 BeginClipArea

```
struct ARGUMENTS {
    DWORD  u32_FuncNo
    DWORD  u32_DcH
    DWORD  u32_Mode;
    DWORD  u32_Control
};
```

This introduces the definition of a clip area, which is terminated by an EndClipArea function. The primitives between these cause no drawing to occur, but instead define a clip area. At EndClipArea, this area is combined, in the manner specified by *u32_mode*, with the existing clip area, to form the new clip area, which is used for subsequent clipping.

It is valid for a normal area (BeginArea .. EndArea) to occur within a clip area definition. If this occurs, the BeginArea and EndArea are effectively ignored, except that, within the area bracket, closure lines are generated as usual. A segment

A null clip area (as, for example, if there are no primitives between the BeginClipArea and the EndClipArea), causes all subsequent drawing to be clipped.

CharString(Pos) functions are only valid for characters drawn with vector symbol sets or outline fonts.

Parameters:-

---

u32_control

A 4-byte parameter containing flags:-

GPICA_WINDING

(bit 1) - Set to '1'B if the clip area is to be constructed in *winding* mode. Otherwise it is constructed in *alternate* mode.

u32_mode

Defines how a new clip area is to be formed from the combination of the old clip area and the one to be defined:-

```
GPICA_UNION          (1) - Union of old and specified areas
GPICA_REPLACE        (2) - Specified area replaces old area
GPICA_SYMDIFF        (4) - Symmetrical difference of specifiec
                           and old areas
GPICA_INTERSECTION   (6) - Intersection of old and specified areas
GPICA_DIFF           (7) - Old area AND NOT(specified area)
GPICA_INFINITE      (17) - New clip area is infinite, regardless
                           of specified area
```

16.1.7.8.5 Function: 030A 0004 EndClipArea

```
struct ARGUMENTS {
    DWORD  u32_FuncNo
    DWORD  u32_DcH
    DWORD  u32_Cancel
};
```

This terminates the definition of a clip area.

BeginClipArea followed by EndClipArea with nothing in between will create a zero sized clip area (that is everything will be clipped away). Vector characters are allowed in a clip area definition. The geometric line thickness attribute is ignored for lines defining a clip area.

Clipping is inclusive at the left and bottom boundaries, and exclusive at the right and top boundaries, (like clip regions). Bounds computation is performed on unclipped primitives. Correlation is performed on the output of primitives that have been clipped to the Viewing Limits and and Graphics Field only (not the clip area).

Parameters:

---

u32_cancel
: If this is 0, the new clip area established, if it is 1, cancel new clip area, leave old clip area unchanged. It is not an error if EndClipArea(Cancel) is issued without BeginClipArea having been issued previously.

16.1.7.8.6 Function: 030A 0005 BeginStrokes

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
};
```

This defines the start of a strokes bracket. The significance of a strokes bracket is that if geometric thick lines are in force, the primitives within the bracket will be drawn as a whole - ie. with line joins rather than line ends between primitives.

16.1.7.8.7 Function: 030A 0006 EndStrokes

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Cancel
};
```

This terminates a strokes bracket.

Parameters:

u32_cancel
: If this is 0, the primitives in the strokes bracket are to be drawn as requested; if it is 1, the primitives in the strokes bracket are discarded. It is not an error if EndStrokes(Cancel) is issued when BeginStrokes has not been called previously.

16.1.7.8.8 Function: 020A 0007 QueryAreaState

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
};
```

This returns whether an area / cliparea / strokes bracket is currently in force.

Returns:

```
0 - not in any area/cliparea/strokes state
1 - in area state
2 - in cliparea state
4 - in strokes state
```

The values are additive if more than one state is current.

16.1.7.8.9 Function: 050A 0008 DrawFrame

```
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 co-ordinates passed are in DC origin device co-ordinates.

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, ie 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
```

16.1.7.9 Bounds Functions - Major Function 0B.

16.1.7.9.1 Function: 030B 0004 QueryCharCorr

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_n
};
```

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.

Parameters:

---

p32_n
: A pointer to the s32 variable in which to return the character offset. A value of zero indicates the first character in the string. A negative value indicates that no string has been correlated on.

16.1.7.9.2 Function: 030B 0005 GetPickWindow

```
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).

16.1.7.9.3 Function: 030B 0006 SetPickWindow

```
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.

16.1.7.10 Clip Functions - Major Function 0C.

Function: 040C 0000 GetClipBox

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Result
    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 and the Xform rectangle.

Note that the Xform rectangle is the intersection of the 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 rectan gle 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.

p32_Result
The complexity of the resultant region from the operation.

0 NULL region
1 RECTanglar region
2 COMPLEX region (more than 1 rectangle)

Function: 050C 0001 SelectClipRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_OldRgnH
    ULONG* p32_Result
    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 clip 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.

Parameters:

u32_RgnH
: The handle of the region. If is null, the clipping region is set to no clipping, its initial state.

p32_Result
: The complexity of the resultant region from the operation.

```
0 NULL region
1 RECTanglar region
2 COMPLEX region (more than 1 rectangle)
```

p32_OldRgnH
: The handle of the previously selected region. A null handle means that there was no clipping.

Function: 040C 0002 IntersectClipRectangle

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Result
    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.

p32_Result
: The complexity of the resultant region from the operation.

```
0 NULL region
1 RECTanglar region
2 COMPLEX region (more than 1 rectangle)
```

Function: 040C 0003 ExcludeClipRectangle

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
```

```
    ULONG* p32_Result
    ULONG* p32_xy
};
```

Excludes the specified rectangle from the clipping region.

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.

p32_Result
The complexity of the resultant region from the operation.

0 NULL region
1 RECTanglar region
2 COMPLEX region (more than 1 rectangle)

Function: 030C 0004 OffsetClipRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Result
    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.

p32_Result
The complexity of the resultant region from the operation.

0 NULL region
1 RECTanglar region
2 COMPLEX region (more than 1 rectangle)

Function: 030C 0005 SetXformRect

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Rect
};
```

This function intersects a rectangle with the existing clipping region of the DC to produce a new clipping region used in subsequent drawing operations.

The clipping region resulting from the intersection is distinct from the current Clip Region which can be set using SelectClipRegion, for example.

Parameters:

p32_Rect
: is a rectangle in device coordinates.

Function: 030C 0006 QueryClipRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
};
```

This returns the handle of the currently selected clip regipn. If there is no currently-selected clip region, NULL is returned.

A return value u32_RegionH is placed in AX:DX:

```
  0 no region selected or error
Non-0 region handle of clip region
```

Function: 040C 0007 PtVisible

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_SuccessVar
    ULONG* p32_xy
};
```

This checks whether a point is visible within the clipping region of the specified device context.

Parameters:

p32_xy Specifies the s32_x, s32_y point in world coordinates.

p32_SuccessVar
: A far pointer to u16_SuccessVar which is set to 1 if the point is visible, and 0 otherwise.

Function: 040C 0008 RectVisible

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_SuccessVar
    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.

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.

P32_SuccessVar
A far pointer to u16_SuccessVar which is set to 2 if the bounding rectangle is totally visible, 1 if it is partially visible and 0 if totally invisible.

Function: 050C 0009 GetClipRects

```
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 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_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_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.

Function: 050C 000A SelectVisRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_OldRgnH
    ULONG* p32_Result
    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 is null, the clipping region is set to no clipping, its initial state.

p32_Result
: The complexity of the resultant region from the operation.

```
0 NULL region
1 RECTanglar region
2 COMPLEX region (more than 1 rectangle)
```

p32_OldRgnH
: The handle of the previously selected region. A null handle means that there was no vis region selected.

Function: 030C 000B QueryVisRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_RgnH
};
```

Returns the handle of the current visible region.

Parameters:

p32_RgnH
: The handle of the returned region

16.1.7.11 Region Functions - Major Function 0D.

16.1.7.11.1 Function: 050D 0000 GetRegionBox

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Result
    ULONG* p32_xy
    ULONG  u32_RgnH
};
```

Returns the dimensions of the tightest rectangle around a 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 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.

p32_Result
: The complexity of the resultant region from the operation.

```
0 null region
1 region of 1 rectangle only
2 complex region (more than 1 rectangle)
```

16.1.7.11.2 Function: 060D 0001 GetRegionRects

```
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 zero value means 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_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.

Note: 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

16.1.7.11.3 Function: 050D 0002 CreateRectRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_count
    ULONG* p32_xy
    ULONG* p32_RgnH
};
```

This creates a region defined using a series of rectangles. The new region is defined by the OR of all the rectangles.

Parameters:

p32_RgnH
: A far pointer to a variable in which the handle of the new region is returned.

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.

16.1.7.11.4 Function: 030D 0003 DestroyRegion

```
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.

16.1.7.11.5 Function: 050D 0004 SetRectRegion

```
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.

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.

16.1.7.11.6 Function: 070D 0005 CombineRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Mode
    ULONG  u32_Src2RgnH
    ULONG  u32_Src1RgnH
    ULONG  u32_DestRgnH
    ULONG* p32_Result
};
```

This combines two regions to make a third.

Parameters:

---

p32_Result
: The complexity of the resultant region from the operation.

  0 NULL region
  1 RECTanglar region
  2 COMPLEX region (more than 1 rectangle)

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 AND
  2 OR
  3 XOR
  4 DIFF
  5 COPY

16.1.7.11.7 Function: 040D 0006 OffsetRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_xy
    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.

16.1.7.11.8 Function: 050D 0007 EqualRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_EqualVar
    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.

p32_EqualVar
: A far pointer to u16_EqualVar which is set to 1 if the two regions are equal, and 0 otherwise.

16.1.7.11.9 Function: 050D 0008 PtInRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_SuccessVar
    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.

p32_SuccessVar
: A far pointer to u16_SuccessVar which is set to 1 if the point is in the region, and 0 otherwise.

16.1.7.11.10 Function: 050D 0009 RectInRegion

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_SuccessVar
    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.

p32_SuccessVar
: A far pointer to u16_SuccessVar which is set to 2 if the rectangle is totally within the region, 1 if it is partially within the region and 0 otherwise.

16.1.7.11.11 Function: 030D 000A PaintRegion

```
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.

16.1.7.12 Transform Functions - Major Function 0E.

16.1.7.12.1 Function: 060E 0000 Convert

```
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.

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.

16.1.7.12.2 Function: 040E 0001 GetModelXform

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_XformData
};
```

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:

  (M11, M12, M21, M22, M41, M42).

16.1.7.12.3 Function: 040E 0002 SetModelXform

```
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:

  (M11, M12, M21, M22, 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`
  2 `Concatenate before`
  3 `Overwrite.`

16.1.7.12.4 Function: 040E 0003 GetWindowViewportXform

```
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:

  (M11, M12, M21, M22, M41, M42),

16.1.7.12.5 Function: 040E 0004 SetWindowViewportXform

```
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.

  (M11, M12, M21, M22, 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
  2 Concatenate before
  3 Overwrite.

16.1.7.12.6 Function: 040E 0006 GetGlobalViewingXform

```
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:

  (M11, M12, M21, M22, M41, M42).

16.1.7.12.7 Function: 040E 0007 SetGlobalViewingXform

```
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.

  (M11, M12, M21, M22, 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
  2 Concatenate before
  3 Overwrite.

16.1.7.12.8 Function: 030E 0008 GetGraphicsField

```
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).

16.1.7.12.9 Function: 030E 0009 SetGraphicsField

```
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).

16.1.7.12.10 Function: 030E 000A GetPageUnits

```
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 page units, height and width are to be stored:

(u32_units, u32_width, u32_height).

16.1.7.12.11 Function: 050E 000B SetPageUnits

```
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'000000' Cells
: Row/Column character cell units, with the origin at (1,1) and y increasing downwards.

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
        (maximised 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 (maximised 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 Cells:

```
Window/Viewport Transform   As defined by Window, Viewport
  where Window is (0,32767) (32767,0)
  and Viewport is (1,1)  (w,h)
Graphics Field               (1,1)  (w,h)
Page Window                  (1,h)  (w,1)
Page Viewport                (0,0)  (Cw*w-1,Ch*h-1)
          where Cw = Cell Width (Normal Default) in pels
```

```
            and   Ch = Cell Height (Normal Default) in pels
Device Transform            As defined by Page Window, Page Viewport
```

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 (maximised window etc.) height in pels.
   Dw is the default device (maximised 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
```

16.1.7.12.12 Function: 030E 000C GetPageWindow

```
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).

16.1.7.12.13 Function: 040E 000D SetPageWindow

```
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.
```

16.1.7.12.14 Function: 030E 000E GetPageViewport

```
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).

16.1.7.12.15 Function: 040E 000F SetPageViewport

```
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 '0'B to indicate that the Device Transform
      should not be modified.
Bits 1-31      Reserved.
```

16.1.7.12.16 Function: 030E 0011 GetDCOrigin

```
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.

16.1.7.12.17 Function: 030E 0012 SetDCOrigin

```
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.

16.1.7.12.18 Function: 030E 0013 GetViewingLimits

```
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).

16.1.7.12.19 Function: 030E 0014 SetViewingLimits

```
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:

If each of the four coordinates is zero, the viewing limits are set to their standard default values.

(s32_x1, s32_y1, s32_x2, s32_y2).

16.1.7.13 Attribute Functions - Major Function 0F.

Function: 030F 0001 EnableKerning

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Flags
};
```

This enables or disables pair and track kerning. The default is that both are disabled when a DC is created.

---

u32_Flags

Denotes whether pair or track kerning are on or off

```
bit 0    - 0 = pair kerning off
           - 1 = pair kerning on

bit 1    - 0 = track kerning off
           - 1 = track kerning on
```

Function: 040F 0002 GetKerningPairTable

```
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

```
struct KERNPAIRS {
       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-characte
                    spacing.
```

Note: The number of kern pairs is a field in the text metrics.

Function: 040F 0003 GetTrackKernTable

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_KernTracks
    ULONG  u32_Count
};
```

Gets the kerning tracks of the current font.

u32_count
: The number of kern tracks the application wants

p32_KernTracks
: A far pointer to an array of kern track records

```
struct KERNTRACK {
       Dword  MinSize    ;
       Dword  MinAmount  ;
       Dword  MaxSize    ;
       Dword  MaxAmount  ;
       }

where  MinSize     4 byte integer indicating minimum
                   font size to which linear tracking
                   applies.
       MinAmount   4 byte integer indicating the amount
                   of inter-character spacing to remove
                   at minimum size.
       MaxSize     4 byte integer indicating maximum
```

```
                                font size to which linear tracking
                                applies.
                    MaxAmount   4 byte integer indicating the amount
                                of inter-character spacing to remove
                                at maximum size.
```

Note: The number of kern tracks in a font is present in the text metrics.

Function: 060F 0004 SetKernTrack

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_Flags
    ULONG* p32_Viewport
};
```

Function: 060F 0005 DeviceSetAttributes

```
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.

Each mask contains a bit corresponding to each attribute in the bundle record, as defined above. For all the valid bits set to *1* in a mask, the corresponding attributes are set to the values indicated.

If both mask bits are set to 1 for a particular attribute then the attribute is set to the value given (not the standard default).

Parameters:

u32_BType
: Specifies the bundle type as one of the following:

```
1   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 set to the values given.

p32_Attrs
: Points to the fixed format bundle record, specified above, containing the attribute values to be set, as specified by *u32AttrsMask* In the record, only the attribute fields which correspond to the attributes to be set contain valid values.

Function: 040F 0006 DeviceSetGlobalAttribute

```
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.

Function: 040F 0007 NotifyClipChange

```
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.

Function: 070F 0008 RealizeFont

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_LogFont
    ULONG  u32_Command
    ULONG  u32_Accelator
    ULONG* p32_EFont
    ULONG* p32_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.

The form of the dialog is:

1. The engine askes the driver if it can realize a device font for the log font requested.
2. The driver answers yes by returning TRUE in ax if it has the font and it fills in the RF_arg_PFont field with a 32 number of the form 0:16.
3. The driver answers no by returning zero in ax.

The engine will then look in its tables attempt to find a font in its tables. If a font is found then the engine calls the display driver with a long pointer to a engine font in RF_arg_EFont. If the driver wishes to use realize that font from the engine font then it returns true in ax and fills in RF_arg_PFont with a 32 number of the form 0:16.

Parameters:

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 Italisizing
TC_UNDERLINE   equ 00100B ; Wants Underlining
TC_DUNDERLINE  equ 00100B ; Wants Double underlining
TC_STRIKEOUT   equ 01000B ; Wants to be StrikeOut
```

p32_PFont
: if(Command == DeviceFont, SymbolSet, LoadEngineFont) The driver fills this in with a 16 bit

identifier if the font or symbol set is realised. The identifier must have the form 0 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.

p32_LogFont
A long pointer to a logical font data structure if command = DeviceFont or a long pointer to a symbol set definition if the command = SymbolSet

u32_Command
A 32 bit command is one of:

DeviceFont
The driver is asked whether it can realize a match.

SymbolSet
The driver is asked whether it can load the symbol set

LoadEngineFont
The driver is ask to use a engine font from the mapper

DeleteFont
The driver is asked to delete a font.

SelectFont
The driver is asked to make this font the current one

Return values:

ax = 1 the device realised/selected/deleted the font

ax = 0 the device was unable to realise/select/delete the font.

Function: 020F 0009 ErasePS

```
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.

Function: 030F 000B GetDCCaps

```
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.

Function: 030F 000C DeviceQueryFontAttributes

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG  u32_AttrsLen
    ULONG* p32_FontAttributes
};
```

This obtains the attributes of the font currently selected via SetCharSet.

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.

```
Familyname          32 byte string
Facename            32 byte string

RegistryId          Word
CodePage            Word
```

```
EmHeight                Word
XHeight                 Word

MaxAscender             Word
MaxDescender            Word
LowerCaseAscent         Word
LowerCaseDescent        Word

InternalLeading         Word
ExternalLeading         Word

AveCharWidth            Word
MaxCharInc              Word

MaxBaselineExt          Word
CharSlope               Word
InlineDir               Word
CharRot                 Word

WeightClass             Word
WidthClass              Word

XDeviceRes              Word
YDeviceRes              Word

FirstChar               Byte
LastChar                Byte
DefaultChar             Byte
BreakChar               Byte

NominalPointSize        Word
MinimumPointSize        Word
MaximumPointSize        Word
TypeFlags               Word
SelectionFlags          Word
Capabilities            Word

SubscriptSize           Word
SubscriptPosition       Word
SuperscriptSize         Word
SuperscriptPosition     Word
UnderscoreWidth         Word
UnderscoreSpacing       Word
StrikeoutSize           Word
StrikeoutPosition       Word
KerningPairs            Word
KerningTracks           Word
Match                   Dword
```

u32_AttrsLen
: The length of the font attributes buffer pointed to by p32_FontAttributes.

Function: 030F 000D DeviceQueryFonts

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_FontCount
```

```
    ULONG  u32_MetricLen
    ULONG* p32_Metrics
    ULONG* p32_FaceName
};
```

This returns a record providing details of the fonts on a device, which match the specified *FaceName*. If the FaceName is null then this is treated as matching all of the fonts in the system.

Parameters:

p32_FaceName
: A far pointer to a null terminated character string specifying the facename. If this is a null pointer then all fonts should be returned.

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 DeviceQueryFontAttributes. 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 far 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.

  Returned value: u32_RemCount

  -1 error
  >=0 The number of fonts not returned. This allows an application to find the number of fonts by specifying a p32_FontCount of zero.

Function: 030F 000F GetPatternOrigin

```
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.

Function: 030F 0010 SetPatternOrigin

```
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 Co-ordinates, 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.

Function: 030F 0011 SetStyleRatio

```
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) nust 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
```

This is a required DDI function for display device drivers.

Parameters:

p32_Ratio
: Points to three 16:16 fixed point numbers. These define the sides of a right angle triangle, which correponds to the aspect ratio of the pels a line is drawn on.

Function: 050F 0012 SetLineTypeGeom

```
struct ARGUMENTS {
    DWORD  u32_FuncNo
    DWORD  u32_DcH
    DWORD* p32_lengths;
    DWORD  u32_count;
    DWORD  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 Co-ordinates, 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
- SetSegmentTransform
- SetModelTransform
- SetWindow
- SetUniformWindow
- SetViewport

- 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_length.

p32_length
: A far pointer to an array, containing *u32_count* elements, which specifies the run lengths, in world co-ordinates. Each array element is of type s32 (long).

Returns: *BOOL*

0 Error
1 OK

Function: 050F 0013 QueryLineTypeGeom

```
struct ARGUMENTS {
    DWORD  u32_FuncNo
    DWORD  u32_DcH
    DWORD* p32_lengths;
    DWORD  u32_count;
    DWORD* 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 pointer 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

16.1.7.14 Color Functions - Major Function 10.

16.1.7.14.1 Function: 0410 0000 QueryColorData

```
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.

Parameters:

---

u32_Count
: The number of elements supplied in *Array.*

p32_Array
: A pointer to, u32_Array, an array which on return contains:-

---

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

Information is only returned for the number of elements supplied. Any extra elements supplied will be zeroed by the system.

16.1.7.14.2 Function: 0610 0001 QueryLogColorTable

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Array
    ULONG* p32_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:-

LOPT_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.

p32_Count
A pointer to the number of elements available in *Array*.

On return, it is updated to the number of elements actually returned. If LOPT_INDEX is specified, only an even number

of elements will be returned.

p32_array
: A pointer to an array in which the information is returned. If LOPT_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 LOPT_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.

16.1.7.14.3 Function: 0710 0002 CreateLogColorTable

```
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 realised.

If this option is not set, RealizeColorTable may have no effect

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
```

16.1.7.14.4 Function: 0210 0003 RealizeColorTable

```
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.

16.1.7.14.5 Function: 0210 0004 UnrealizeColorTable

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
};
```

This function is the reverse of RealizeColorTable. It causes the default color table to be reinstated.

16.1.7.14.6 Function: 0610 0005 QueryRealColors

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Array
    ULONG* p32_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:-

  LOPT_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.

  LOPT_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.

p32_Count
: A pointer to the number of elements available in *array*. On return this is updated to the number of elements actually returned. If LOPT_INDEX is specified, only an even number of elements will be returned.

p32_Array
: A pointer to a u32_ array in which the information is returned. If LOPT_INDEX = B'0', this is an array of color values (each value is as defined for CreateLogColorTable). If LOPT_INDEX = B'1', it is an array of alternating color indices and values, in the order index1, value1, index2, value2,...

16.1.7.14.7 Function: 0510 0006 QueryNearestColor

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_RgbColorOut
    ULONG  u32_RgbColorIn
    ULONG  u32_Options
};
```

Returns the nearest color available to the specified color, on the currently associated device. Both colors are specified in RGB terms.

Parameters:

---

u32_Options
: Specifies various options:-

  LOPT_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

p32_RgbColorOut
: A pointer to *U32_RgbColorOut* containing the nearest available color to the specified color.

16.1.7.14.8 Function: 0510 0007 QueryColorIndex

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_Color
    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.

Parameters:

---

u32_Options (ULONG)
: Specifies various options:-

LOPT_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

p32_Color
A pointer to a variable in which the closest match color index is returned.

16.1.7.14.9 Function: 0510 0008 QueryRGBColor

```
struct ARGUMENTS {
    ULONG  u32_FuncNo
    ULONG  u32_DcH
    ULONG* p32_RgbColor
    ULONG  u32_Color
    ULONG  u32_Options
};
```

This returns the actual RGB color which will result from the specified color index, for the device specified.

Parameters:

u32_Options
Specifies various options:-

LOPT_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

p32_RgbColor
A pointer to a variable in which the corresponding RGB color is returned.

16.1.7.15 Query Functions - Major Function 11.

16.1.7.15.1 Function: 0411 0000 QueryDeviceBitmaps

```
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).

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.

16.1.7.15.2 Function: 0411 0001 QueryDeviceCaps

```
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 following element numbers are defined:-

1	Device family (values as for u32_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 (eg 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 Vio

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 GpiBitBlt supported
Bit 1 - 1 if this device supports banding
Bit 2 - 1 if GpiBitBlt with scaling supported
Bit 3 - 1 if GpiFloodFill supported
Bit 4 - 1 if GpiSetPel supported

26 Default marker box width in pels

27 Default marker box depth in pels

28 Number of device specific fonts

29 Reserved (for graphics drawing subset supported)

30 Reserved (for graphics architecture version number supported)

31 Reserved (for 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:

-1 Error
0 OK

16.1.7.15.3 Function: 0711 0003 Escape

```
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 made by an application are translated and sent to the device driver.

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
- 11-32767 - Reserved

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.

u32_OutCount
: Specifies the size of the buffer pointed to by p32_Outdata.

p32_OutData
: A far pointer to the data structure to receive data from this escape.

Returns:

```
-1 Error
 0 Escape not implemented for specified code
 1 OK
```

16.1.7.15.4 Function: 0511 0004 QueryHardcopyCaps

```
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 form code number the query is to start from. 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 (ie 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 formname[32];
     LONG xwidth;
```

```
    LONG yheight;
    LONG xleftclip;
    LONG ybottomclip;
    LONG xrightclip;
    LONG ytopclip;
    LONG xpels;
    LONG ypels;
   ;
```

---

formname
: The ASCIIZ name of the form.

xwidth
: The width (left to right) in millimeters.

yheight
: 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.

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:

-1 Error
>=0 If *count* == 0, number of forms available
>=0 If *count* != 0, number of forms returned

16.1.7.16 Device Modes Function - Major Function 14.

16.1.7.16.1 Function: 0614 0005 DeviceMode*

```
struct ARGUMENTS {
    ULONG   u32_FuncNo;
    ULONG   u32_DcH
    ULONG*  p32_LogAddr
    ULONG*  p32_DeviceName;
    ULONG*  p32_DriverName;
    ULONG*  p32_DriverData;
```

```
};
```

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 functioncna 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 (eg "LPT1").

Returns:

```
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
```

16.1.8 Graphics Engine Functions Callable by Device Drivers.

The Graphics Engine provides a number of functions which can be used by device drivers to assist in their work. This section provides a list of these functions.

16.1.8.1 Brief List of DDI-Engine Function Calls

- AccumulateBounds
- GreGetCodepageTable
- GreGetRevCodeTable
- ClipPoly
- ClipLine
- ClipConic
- ClipRect
- ClipScans

16.1.8.2 Description of DDI-Engine Function Calls

AccumulateBounds (lpArgs,Command,hDC,FunN)

```
ULONG   *lpArgs;
ULONG    Command;
HANDLE   hDC;
ULONG    FunN;
```

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

The lpArgs parameter points to an argument structure as

follows:

```
struct ARGUMENTS {
    ULONG   FunN;
    ULONG   DcH;
    ULONG*  p32_lpRect
};
```

p32_lpRect 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.

GreGetCodepageTable (s32_cpid, p32_tab, u32_FuncNo)

```
ULONG  u32_FuncNo
ULONG* p32_Tab
ULONG  s32_CpID
```

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

GreGetRevCodeTable (s32_cpid, p32_tab, u32_FuncNo)

```
ULONG  u32_FuncNo
ULONG* p32_Tab
ULONG  s32_CpID
```

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

ClipPoly (p32_CallBack, u32_DCH, u32_FunN)

```
ULONG* p32_CallBack
```

This is an engine call to provide the driver with the current clip polygon.

This allows devices which do not perform boolean operations of clip areas, but do clip to areas, to query the result of the boolean area combining.

Parameters:

p32_CallBack
: is a function with the same form as a SimulationEntry

  ```
  CallBack (lpparm, command, hDC, FunN)
  ```

  The function will be called back with three different primitives:

  1. SetCurrentPosition, this denotes the beginning of a subarea
  2. Polyline
  3. PolyFilletSharp

The subareas are "simple", ie non self intersecting (or with other subareas)

The callback function may be terminated at any time by returning zero to the engine.

ClipLine (p32_XY, s32_Count, p32_CallBack, u32_DCH, u32_FunN)

```
ULONG* p32_CallBack
```

```
ULONG  s32_Count
ULONG* p32_XY
```

This function may be 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 and function numbers are the same as for PolyLine, and the callback has the same form as the other DDI routines.

The callback function may be terminated at any time by returning zero to the engine.

ClipConic
(p32_XY,s32_Count,p32_Sharp,p32_CallBack,u32_DCH,u32_FunN)

```
ULONG* p32_CallBack
ULONG* p32_Sharp
ULONG  s32_Count
ULONG* p32_XY
```

This function may be 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 and function numbers are the same as for PolyFilletSharp, and the callback has the same form as the other DDI routines.

The callback function may be terminated at any time by returning zero to the engine.

ClipRect (p32_Rect, p32_CallBack, u32_DCH, u32_FunN)

```
ULONG* p32_CallBack
ULONG* p32_Rect
```

This function may be 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 and function numbers are the same as for Box, and the callback has the same form as the other DDI routines.

The callback function may be terminated at any time by returning zero to the engine.

ClipScans
(p32_psl1,p32_psl2,p32_BoundingRect,p32_CallBack,u32_DCH,u32_FunN)

```
ULONG* p32_CallBack
ULONG* p32_BoundingRect
ULONG* p32_psl2
```

```
ULONG* p32_psll
```

This function may be used by the driver to pass a polyscanline to the engine. The engine will clip it against the clip region, and call the callback function with a clipped polyscanline.

The parameters and function number are the same as for PolyScanLine, and the callback has the same form as PolyScanLine.

The callback function may be terminated at any time by returning zero to the engine.

16.1.9 Required Functions

16.1.9.1 All Devices

All device drivers must support the following functions:

- PolyLine
- PolyShortLine
- LineDDA
- GetCurrentPosition
- SetCurrentPosition
- ScanLR
- PolyScanline
- GetBitmapParameters
- GetBitmapBits
- SetBitmapBits
- GetPixel
- SetPixel
- ImageData
- BitBlt
- CharStringPos
- CharStr
- CharRect
- ScrollRect

- UpdateCursor
- DisableKerning
- EnableKerning
- GetKerningPairTable
- GetTrackKernTable
- GetAttributes
- SetAttributes
- SetSingularAttribute
- SetGlobalAttribute
- GetRelWidths
- SetRelWidths
- GetColorIndex
- GetRealColors
- LoadLogColorTable
- SelectLogColorTable
- RealizeColorTable
- UnrealizeColorTable
- QueryColors
- QueryNearestColor
- QueryDeviceBitmaps
- EmunFonts
- QueryTextMetrics
- GetExtentTable
- QueryTextBox
- QueryTextBreak
- RealizeFont
- Enable
- Disable
- DeviceMode
- Escape

The remaining functions are all optional, the driver may hook as many of them as it wants.

16.1.9.2 Display Devices

If the device is the main display device, it must have the following (this includes support for the mouse cursor);

- Inquire
- CheckCursor
- SetCursor
- MoveCursor
- DrawFrame
- SetStyleRatio

16.1.9.3 Printer Devices

A printer device driver must interact with the Spooler, and meet its requirements.

16.1.10 Clipping

The driver is responsible for clipping its output according to the active clip region. The function, GetClipRectangles, is provided by the graphics engine to let the device driver examine all the rectangles making up the active clip region.

The device driver could ask for an enumeration of all the rectangles in the clip region each time it is called on to draw. Alternatively, the engine provides a function, NotifyClipChange, which can be used to get notification when the active clip region is changed. This function may be useful to drivers that wish to maintain their own information about the clip region.

Appendixes

Appendix A
Font File Format

A.1 Introduction

The Presentation Manager 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 descibed in the following sections.

The font file is a set of self defining records, where the records have the form:

```
font_record struc
    frIdentity   dd    ?
    frSize       dd    ?
    frBytes      db    bytes of information
font_record struc
```

There are four records in a font resource:

- The font metrics
- The font character definitions
- The pair kerning table
- The track kerning table

Following compilation, the records in the resource file will follow one another in the above order.

A.2 Font Metrics

The following is a list of all the Font Attributes, in the order in which they occur in the font file.

This information appears in the font directory in the resource file.

- Identity
- Size
- Family name
- Typeface name
- Registry ID
- Code page

- Em Height
- 'x' Height
- Maximum Ascender
- Maximum Descender
- Lower Case Ascent
- Lower Case Descent
- Internal Leading
- External Leading
- Average Character Width
- Maximum Character Increment
- Maximum Baseline Extent
- Character Slope
- Inline Direction
- Character Rotation
- Weight Class
- Width Class
- X Device Resolution
- Y Device Resolution
- First character
- Last character
- Default Character Code Point
- Break Character
- Nominal Point Size
- Minimum Point Size
- Maximum Point Size
- Type Flags (incl. Fixed/Variable Pitch)
- Selection Flags - Italic-Ness, Underline-Ness, Strikeout-Ness.
- Capabilities
- Subscript Size
- Subscript Position
- Superscript Size
- Superscript Position

- Underscore Size
- Underscore Position
- Strikeout Size
- Strikeout Position
- Kerning Pairs
- Kerning Tracks
- Device Name Offset

The Font Attributes are described in detail in the following section.

Identity 4 byte integer

Size 4 byte integer

Family Name
FACESIZE (32) character string

The family name of the font, the basic appearance of the font eg. Bodoni. This will be used when the application finds a font with a facename such as Bondoni Italic on a printer. If no such font is available on the screen, then the application can see whether another font of the same basic form is available for use.

Face Name
32 character string

The typeface name to which the font is designed, eg. Times Roman.

Registry ID
2 byte integer

The Registry number for the font

Code Page
2 byte integer

Defines the Registered Code Page supported by the font.

Em Height
2 byte integer.

The (average) height above the baseline for uppercase characters.

'x' Height
2 byte integer.

The (average) height above the baseline for lowercase characters.

Maximum Ascender
2 byte integer

The maximum height above the baseline reached by any part of any symbol in the font.

Maximum Descender
2 byte integer

The maximum depth below the baseline reached by any part of any symbol in the font.

Lower Case Ascent
2 byte integer

The maximum height above the baseline reached by any part of any lower case symbol in the font.

Lower Case Descent
2 byte integer

The maximum depth below the baseline reached by any part of any lower case symbol in the font.

Internal Leading
2 byte integer

Recommended External Leading
2 byte integer

Average Character Width
2 byte integer

Average inter-character increment for the font. For a more detailed description, see the section, "Average Character Width – Definition Formula".

Maximum Character Increment
2 byte integer.

The maximum inter-character increment for the font.

Maximum Baseline Extent
2 byte integer.

This is essentially the vertical space required by the font - ie the nominal inter-line gap.

Character Slope
2 byte integer.

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.

Inline Direction
2 byte integer

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.

Inline direction, like other rotations, is represented by a two part unsigned discontinuous quantity. The first 9 bits constitute a value in the range 0 to 359, representing the number of degrees in the rotation. The next 6 bits constitute a number in the range 0 to 59, representing the number of minutes in the rotation. The final bit is reserved 0. Values outside the specified ranges are invalid.

Character Rotation

2 byte integer

The baseline direction for which the characters in the font are designed.

Weight Class

2 byte integer.

Indicates the visual weight (thickness of strokes) of the characters in the font:

Value	Description
1	Ultra-light
2	Extra-light
3	Light
4	Semi-light
5	Medium (normal)
6	Semi-bold
7	Bold
8	Extra-bold
9	Ultra-bold

Width Class

2 byte integer.

Indicates the relative aspect ratio of the characters of the font in relation to the 'normal' aspect ratio for this type of font:

Value	Description	% of Normal
1	Ultra-condensed	50
2	Extra-condensed	62.5
3	Condensed	75
4	Semi-condensed	87.5
5	Medium (normal)	100
6	Semi-expanded	112.5
7	Expanded	125
8	Extra-expanded	150
9	Ultra-expanded	200

Target Device Resolution - X

2 byte integer

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
2 byte integer

The resolution in the y dimension of the device.

First Character
1 byte integer

The code point of the first character in the font.

Last Character
1 byte integer

The code point of the last character in the font.

All code points between the first and last character specified must be supported by the font.

Default Character Code Point
1 byte integer

The code point which is used if a code point outside the range supported by the font is used.

Break Character
1 byte integer

The code point which represents the 'space' or 'break' character for this font.

Nominal Vertical Point Size
2 byte integer

The height of the font specified in decipoints (one 720th of an inch). The nominal size is the size for which the font is designed.

Minimum Vertical Point Size
2 byte integer.

The mimimum height to which the font may be scaled down for display.

Maximum Vertical Point Size
2 byte integer.

The maximum height to which the font may be scaled up for display.

Type Flags
2 bytes of bits.

Contains the following information:

- Bits 0-4 Reserved 0
- Bit 5 - Font has Kerning information (1=has kerning data)
- Bit 6 - Fixed/Proportional Spaced Font (1=Fixed)
- Bit 7 - Protected(Licensed) Font

Selection Flags
2 bytes of bits.

Contain information concerning the nature of the font patterns, as follows:

```
Bit Number  Meaning
     0      0 = Upright graphic characters
            1 = Italic graphic characters
     1      0 = Graphic characters are not underscored
            1 = Graphic characters are underscored
     2      0 = Positive Image characters
            1 = Negative Image characters
     3      0 = Solid graphic characters
            1 = Outline (hollow) graphic characters
     4      0 = Graphic characters are not overstruck
            1 = Graphic characters are overstruck
  5-15      Reserved zeros
```

Capabilities
2 byte integer

This tells the engine what sort of simulations can be done on the font and are basically to do with sizing and synthesis.

Recommended Subscript Size
2 byte integer

The recommended point size for Subscripts for this font.

Recommended Subscript Position
2 byte integer

The recommended baseline offset for Subscripts for this font.

Recommended Superscript Size
2 byte integer

The recommended point size for Superscripts for this font.

Recommended Superscript Position
2 byte integer

The recommended baseline offset for Superscripts for this font.

Underscore Size
1 byte integer

The number of strokes used to underscore the characters of the font.

Underscore Position
: 2 byte integer.

  The position of the (first) underscore stroke from the baseline.

Strikeout Size
: 2 byte integer

  Thickness of the overstrike stroke.

Strikeout Position
: 2 byte integer

  The position of the overstrike stroke relative to the baseline.

Kerning Pairs
: 2 byte integer

  The number of kerning pairs.

Kerning Tracks
: 2 byte integer

  The number of kerning tracks.

Device Name Offset
: 4 byte integer

  This is an offset from the beginning of the resource to a null terminated string with the name of the device, eg. Epson FX-80. The mapper can tell if this font belongs to a specific device. If a font is generic, this field should be zero.

A.2.1 Font Attributes Layout in Storage.

The storage layout of the Font Attributes is as follows:

```
FONTMETRICS     struc
    tmIdentity              dd  ?       ; must be equal to 1
    tmSize                  dd  ?

    tmFamilyname            db  FACESIZE dup (?)
    tmFacename              db  FACESIZE dup (?)

    tmRegistryId            dw  ?
    tmCodePage              dw  ?

    tmEmHeight              dw  ?
    tmXHeight               dw  ?

    tmMaxAscender           dw  ?
    tmMaxDescender          dw  ?
    tmLowerCaseAscent       dw  ?
    tmLowerCaseDescent      dw  ?
```

```
        tmInternalLeading       dw  ?
        tmExternalLeading       dw  ?

        tmAveCharWidth          dw  ?
        tmMaxCharInc            dw  ?

        tmMaxBaselineExt        dw  ?
        tmCharSlope             dw  ?
        tmInlineDir             dw  ?
        tmCharRot               dw  ?

        tmWeightClass           dw  ?
        tmWidthClass            dw  ?

        tmXDeviceRes            dw  ?
        tmYDeviceRes            dw  ?

        tmFirstChar             db  ?
        tmLastChar              db  ?
        tmDefaultChar           db  ?
        tmBreakChar             db  ?

        tmNominalPointSize      dw  ?
        tmMinimumPointSize      dw  ?
        tmMaximumPointSize      dw  ?
        tmTypeFlags             dw  ?
        tmSelectionFlags        dw  ?
        tmCapabilities          dw  ?

        tmSubscriptSize         dw  ?
        tmSubscriptPosition     dw  ?

        tmSuperscriptSize       dw  ?
        tmSuperscriptPosition   dw  ?

        tmUnderscoreSize        dw  ?
        tmUnderscorePosition    dw  ?

        tmStrikeoutSize         dw  ?
        tmStrikeoutPosition     dw  ?

        tmKerningPairs          dw  ?
        tmKerningTracks         dw  ?

        tmDeviceNameOffset      dd  ?

FONTMETRICS ends
```

A.2.2 Average Character Width - Definition Formula.

The Average Character Width is calculated according to the following formula:

For the lower case letters ONLY, sum the individual character widths multiplied by the following weighting factors and then divide by 1000:

Letter	Weight Factor
a	64
b	14
c	27
d	35
e	100
f	20
g	14
h	42
i	63
j	3
k	6
l	35
m	20
n	56
o	56
p	17
q	4
r	49
s	56
t	71
u	31
v	10
w	18
x	3
y	18
z	2
space	166

A.3 Font Character Definitions.

Two formats of Font Character definition are supported. These are:

- Image format.

  The Character Glyphs are represented as Pixel images.

- Outline format.

  The Character Glyphs are represented by vector data which traces the outline of the character.

The definition consists of a header portion and a portion carrying the characters themselves.

The header portion contains information about the format of the character definitions and data about each character including width data and the offset into the definition section at which the character definition begins.

There are many possible features to the file format. Presentation Manager will support those listed below. In all representations listed here, the height of all characters in the file is the same.

1. Fixed pitch a+b+c = character increment for all characters

   ```
   a,b,c > 0
   ```

2. Proportional characters a+b+c = character increment for each character

   ```
   a,b,c >= 0
   ```

3. Characters where a,b,c are definitions for all characters

   ```
   b >= 0
   a, c any integer
   ```

A.3.1 Font Definition Header.

Font Definition Identity
: 4 bytes

  Must be equal to 2

Font Definition Size
: 4 bytes

Font Definition Type
: Indicates which format of font definition follows:

  - Bit 0
    - 0 = Image Font Definition
    - 1 = Outline (Vector) Font Definition
  - Bit 1
    - 0 = Device
    - 1 = Generic

Font Definition Fields Flags
: 2 bytes of flags

  Indicates which fields are present in the Font Definition Data in the header.

Bit Number	Meaning
0	0 = Cell Width not defined in header 1 = Cell Width defined in header
1	Cell Height defined in header
2	Character Increment defined in header
3	Character a space defined in header
4	Character b space defined in header
5	Character c space defined in header
6	Baseline offset defined in header
7-15	Reserved zeros

See the section, "Definition Fields Flags Examples", for an illustration of how the font definition field flags are used.

Font Character Definition Fields Flags
2 bytes of flags

Indicates which fields are present on a per character basis.

Bit Number	Meaning
0	0 = Cell Width not defined for each character 1 = Cell Width defined for each character
1	Cell Height defined for each character
2	Character Increment defined for each character
3	Character a space defined for each character
4	Character b space defined for each character
5	Character c space defined for each character
6	Baseline offset defined for each character
7	Character definition offset defined for each character
8-15	Reserved zeros

See the section, "Definition Fields Flags Examples", for an illustration of how the font character definition field flags are used.

Size of Per Character Definition Record
2 byte integer

Indicates the length in bytes of the Character Definition Record (the per character data) in the Font Definition Header.

Character Cell Width
2 byte integer

The width of the characters in pels.

Character Cell Height
2 byte integer

The height of the characters in pels.

Character Increment
2 byte integer

The length along the character baseline required to step from one character to the next (when forming a character string).

Character a Space
2 byte signed integer

The width of the space before a character in the inline direction.

Character b Space
2 byte integer

The width of a character (inline direction).

Character c Space
2 byte signed integer

The width of the space after a character in the inline direction.

Character Baseline Offset
2 byte signed integer

The position of the bottom of a character definition relative to the the baseline in the direction perpendicular to the baseline.

Character Definition Record
n byte record

The following fields may or may not be present, according to the Font Character Definition Fields Flags. If a field is present, then it is present for EACH character and the value applies to that character only.

- Character Definition Offset - 2 byte integer

  The offset into the Font Definition data at which the character definition begins.

- Character Cell Width - 2 byte integer

  The width of the character definition in pels.

- Character Cell Height - 2 byte integer

  The height of the character definition in pels.

- Character Increment - 2 byte integer

  The length along the character baseline required to step from this character to the next (when forming a character string).

- Character a Space - 2 byte signed integer

  The width of the space before the character in the inline direction.

- Character b Space - 2 byte integer

  The width of the character (inline direction).

- Character c Space - 2 byte signed integer

  The width of the space after the character in the inline direction.

- Character Baseline Offset - 2 byte signed integer

  The position of the bottom of the character definition relative to the the baseline in the direction perpendicular to the baseline.

A.3.1.1 Definition Fields Flags Examples

The following illustrates how the Font Definition Fields Flags are used for the representations given in the section, "Font Character Definitions".

- For Fixed pitch where a+b+c = character increment for all characters

  Font Definition Fields Flags

```
Bit 0 = 1 Cell width defined in header, tmAveCharWidth = tmMaxCharI
Bit 1 = 0
Bit 2 = 1 This is valid since all character increments are the same

Bit 3 = 0 A space not defined
Bit 4 = 0 B space not defined
Bit 5 = 0 C space not defined
Bit 6 = 0
```

  Font Character Definition Fields Flags

```
All bits, except 7, are zero
```

- For Proportional characters where a+b+c = character increment for each character

  Font Definition Fields Flags

```
Bit 0 = 0 Cell width not defined in header
Bit 1 = 0
Bit 2 = 0 char increment not defined in header
Bit 3 = 0 A space not defined
Bit 4 = 0 B space not defined
Bit 5 = 0 C space not defined
Bit 6 = 0
```

  Font Character Definition Fields Flags

```
Bit 0 = 1 cell width defined for each character
Bit 1 = 0 height is the same for all characters
Bit 2 = 0 character increment equals the cell width
Bit 3 = 0 A space not defined
Bit 4 = 0 B space not defined
Bit 5 = 0 C space not defined
Bit 6 = 0
Bit 7 = 1 character definition offset defined for each character
```

- For characters where a,b,c is specified for each character

  Font Definition Fields Flags

```
Bit 0 = 0 Cell width not defined in header
Bit 1 = 0
Bit 2 = 0 char increment not defined in header
Bit 3 = 0 A space not defined
Bit 4 = 0 B space not defined
Bit 5 = 0 C space not defined
Bit 6 = 0
```

  Font Character Definition Fields Flags

```
Bit 0 = 0 cell width defined for each character
Bit 1 = 0 height is the same for all characters
Bit 2 = 0 character increment equals a+b+c
Bit 3 = 1 A space defined
Bit 4 = 1 B space defined
Bit 5 = 1 C space defined
Bit 6 = 0
Bit 7 = 1 character definition offset defined for each character
```

A.3.2 Definition Data.

Since for these formats the characters offset is always present, the definitions themselves can be anywhere in the file.

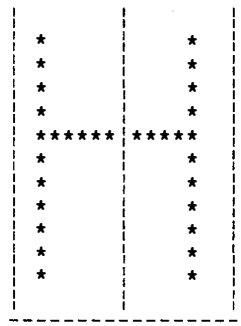
The Definition Data consists of:

Identity	2 byte integer	
Definition Length	4 byte integer	Length of Data in bytes, including the length field.
Outline data	n bytes	The data of the outlines.

A.3.2.1 Image Data format.

The bits for each character are stored separately and start on a byte boundary. Sequential bytes represent vertical pieces of the character image. For example a 15 bit wide H would be stored as follows

```
   1       2
----------------
```



The bytes for section one are all stored sequentially and all of each byte in used This takes up a number of bytes equal to the height of the character.

section 2 are the bytes for the next column of the character. These masks use only seven bits of each byte. Each seven bits is in a different byte.

Thus the character is laid down in byte wide columns.

A.3.2.2 Outline Data format.

The Outline format for character definitions is a set of drawing orders for each characters. The drawing orders permitted within the definitions are:

- Line - GLINE, GCLINE
- Relative Line - GRLINE, GCRLINE
- Begin Area - GBAR
- End Area - GEAR
- Fillet - GFLT, GCFLT
- Set Colour - GSCOL, GSECOL
- Set Line Type - GSLT
- Set Line Width - GSLW
- End Symbol Definition - GESD

A.4 The Pair Kerning Table

There are two possible kerning records. The presence of these records can be detected from the tmKerningPairs and tmKerningTracks fields in the font header

```
Pair_kern_table struc
    pktIdentity     dd  ?   ; must be equal to 3
    pktSize         dd  ?
    pktPairs        db  SIZE kern_pairs     ; an array of kptCount kern pai
Pair_kern_table ends

Kern_pairs  struc
    kpChar1      db ?
    kpChar2      db ?
    kpKernAmount dw ?
```

```
kern_pairs ends
```

Only the data for pairs of characters which can be kerned is provided. The format of the data for each Kern Pair is:

- Char1

  Code Point for first character

- Char2

  Code Point for second character

- Kern Amount

  2 byte signed integer indicating amount of kerning, with positive numbers meaning increase inter-character spacing.

A.5 The Track Kerning Table

```
track_kern_table struc
    pttIdentity     dd  ?   ; must be equal to 4
    pttSize         dd  ?
    ptttracks       db  SIZE kern_tracks    ; an array of kptCount kern tra
track_kern_table ends

Kern_track  struc
    ktMinSize   dd ?
    ktMinAmount dd ?
    ktMaxSize   dd ?
    ktMinAmount dd ?
kern_track ends
```

The model for the dynamic kerning data is one of a linear amount of whitespace reduction between a minimum and a maximum font size. The following fields define the tracks:

1. Minimum Size

   4 byte integer indicating minimum font size to which linear tracking applies.

2. Minimum Amount

   4 byte integer indicating the amount of inter- character spacing to remove at Minimum Size.

3. Maximum Size

   4 byte integer indicating maximum font size to which linear tracking applies.

4. Maximum Amount

   4 byte integer indicating the amount of inter- character spacing to

remove at Maximum Size.

A.6 Presentation Manager Multi-Codepage Font Support.

Presentation Manager supports multiple Codepages for text input and output. For graphics text output using Fonts, a single Font Resource is used to support all the Codepages. Thus the fonts are Multi-Codepage. The following section describes how this function is provided and gives details of the font resource format.

A.6.1 Font Codepage Functions.

Presentation Manager supports the following Codepages for graphics text output:

500	EBCDIC CECP International version.
437	Original PC ASCII codepage.
850	New PC ASCII codepage supporting US English and many European languages.
860	PC ASCII for Portuguese.
863	PC ASCII for Canadian French.
865	PC ASCII for Nordic languages.

Most of the characters required by each codepage are common - for example, the first 128 characters of all the ASCII codepages are identical. This makes it possible, indeed highly desirable, for a single font file definition to support all the codepages - ie a multi-codepage font. Such a font contains an ordered list of ALL the character definitions ('glyphs') used by the collection of codepages above.

To use such a multi-codepage font, all that is required is a mapping from the codepoints of the current codepage to the glyphs of the font. Such a mapping is provided for each of the codepages. To facilitate translation of text strings from codepage to codepage, a mapping from the 'universal' set of characters to each codepage is also provided.

The ordering of the characters is the same in all multi-codepage fonts so that only one set of translate tables is necessary.

A.6.2 Multi-Codepage Font layout.

The ordering of characters in the multi-codepage fonts is based on that of codepage 850, with additional characters added beyond the 256th to provide those characters not present in codepage 850.

This makes mapping codepage 850 into the multi-codepage fonts simple. It also provides simple mappings for the first 128 characters of all the ASCII codepages.

The following extra glyphs are added to codepage 850. They are shown in the order they occur in the multi-codepage font, starting at character number 256:

Index Number	Glyph ID	Colloquial Name
256	SC040000	Cent sign
257	SC050000	Yen sign
258	SC060000	Pesetas sign
259	SM680000	Left-hand not sign
260	SF190000	Double line join single vertical
261	SF200000	Single line join double vertical
262	SF210000	Single line top right corner double
263	SF220000	Double line top right corner single
264	SF270000	Single line bottom right corner double
265	SF280000	Double line bottom right corner single
266	SF360000	Single vertical join double line
267	SF370000	Double vertical join single line
268	SF450000	Double horizontal join single line above
269	SF460000	Single horizontal join double line above
270	SF470000	Double horizontal join single line below
271	SF480000	Single horizontal join double line below
275	SF490000	Double line bottom left corner single
274	SF500000	Single line bottom left corner double
273	SF510000	Single line top left corner double
272	SF520000	Double line top left corner single
273	SF530000	Double vertical cross single
274	SF540000	Single vertical cross double
275	SF580000	Left hand half-block
276	SF590000	Right hand half-block
277	GA010000	Greek alpha lower case
278	GG020000	Greek gamma upper case
279	GP010000	Greek pi lower case
280	GS020000	Greek sigma upper case
281	GS010000	Greek sigma lower case
282	GT010000	Greek tau lower case
283	GF020000	Greek phi upper case
284	GT620000	Greek theta upper case
285	G0320000	Greek omega upper case
286	GD010000	Greek delta lower case
287	SA450000	Infinity sign
288	GF010001	Greek phi lower case
289	GE010000	Greek epsilon lower case

```
290     SA380000 Mathematical intersection sign
291     SA480000 Mathematical equivalence sign
292     SA530000 Mathematical greater than or equals sign
293     SA520000 Mathematical less than or equals sign
294     SS260000 Mathematical integral sign top half
295     SS270000 Mathematical integral sign bottom half
296     SA700000 Mathematical approximately equals sign
297     SA790000 Mathematical product dot
298     SA800000 Mathematical square root sign
299     LNO11000 Superscript small n
```

Thus the multi-codepage fonts have 300 characters in all (including the NULL character).

This number of characters is supported by the font format definition for both image and vector (outline) fonts.

Appendix B
Migration and Coexistence

B.1 Running MS OS/2 applications under Presentation Manager

It is desirable, and possible, that many applications written to run in their own screen group under MS OS/2 should be able to run in a window in the Presentation Manager screen group. However, there are some applications, mainly those that exercise the more specific forms of control of screen and input devices, that will not run. The MS OS/2 facilities that are incompatible with running under Presentation Manager are listed here.

B.1.1 Real mode applications

No real mode application may run in the Presentation Manager screen group.

B.1.2 Device control applications

No application that provides its own SCRN, KBD, or MOU device driver may run, unless that device driver is compatible to the MS OS/2 device driver.

No application that provides its own MOUSCRN device driver may run.

No application that issues IOCTLS to any of the above device drivers may run.

B.1.3 Unsupported VIO calls

The following calls are either unsupported or have restricted function in the Presentation Manager screen group. For further details refer to the chapter, "Standard Application Support".

VioDeRegister
: Deregister a video subsystem

VioGetPhysBuf
: Return the address of the physical video buffer

VioGetState
: Return the current setting of the video state

VioModeUndo
: Restore mode undo

VioModeWait
: Restore mode wait

VioRegister
: Register a video subsystem within a screen group

VioSavRedrawWait
: Screen save redraw wait.

VioSavRedrawUndo
: Screen save redraw undo.

VioScrLock
: lock screen.

VioScrUnlock
: Unlock screen.

VioSetState
: Set the video state.

B.1.4 Unsupported KBD calls

The following calls are either unsupported or have restricted function in the Presentation Manager screen group. For further details refer to the chapter, "Standard Application Support".

- KbdRegister
- KbdDeRegister
- KbdSetStatus
- KbdGetStatus

B.1.5 Unsupported MOU calls

The following calls are either unsupported or have restricted function in the Presentation Manager screen group. For further details refer to the chapter, "Standard Application Support".

- MouRegister
- MouDeRegister
- MouSetHotKey
- MouDrawPtr
- MouRemovePtr
- MouSetPtrPos

— MouSetScaleFact
— MouOpen
— MouClose
— MouGetPtrPos
— MouGetPtrShape
— MouSetPtrShape
— MouGetDevStatus
— MouSetHotKey

B.2 Migration from Microsoft Windows

Microsoft Windows version 1 applications will have to be rewritten to run under Presentation Manager. The following areas will require to be changed:

Window functions
: The windowing and input functions of Presentation Manager are functionally similar to the object oriented message interface of Microsoft Windows, although the function names, messages, and associated data structures are different.

GDI functions
: The graphics functions of Presentation Manager are based on the graphics interfaces found in GPI. This has been extended to include the functionality of the GDI.

System resource functions
: The system resource functions are to a large extent provided by MS OS/2 kernel. However, Presentation Manager provides additional functions equivalent to Microsoft Windows in the following areas:

  — Memory manager functions
  — Resource manager functions
  — String translation functions
  — Atom manager functions

Index