Microsoft®
Operating
System/2

Windows Presentation Manager Reference

Volume 2

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 5
Input Functions

5.1 Input functions

5.1.1 Message Manager Architecture

Every window in the system has a procedure associated with it called the Window Procedure. Communication with windows is done with "window messages", which are sent to the window proc of a window.

The arguments to the window proc make up a window message. There are four parts to a window msg, which correspond to the 4 arguments of a window procedure:

```
HWND   hwnd       - Handle of window receiving the message
UINT   msg        - msg ID identifying the message
ULONG  lParam1    - ULONG parameter (content depends on message ID)
ULONG  lParam2    - ULONG parameter (content depends on message ID)
```

A window message also has a ULONG return value.

The message ID defines the message. The contents of lParam1 and lParam2, and whether or not a return value is required, depend on the semantics of the message as defined by the message ID.

The names of the predefined message IDs begin with "WM_". There are two ways that you can define your own messages: you can either use an integer constant within a certain range, or you can use the atom manager functions to define a window message whose value is unique across all windows in the system. The atom manager need only be used when the same message must be understood by more than one application. For sending private messages within an application, you can use any integer constant in the range WM_ USER to 0x7fff. Values 0x8000 thru 0xbfff are reserved for use by the system.

The lParam2 parameter often contains more than one piece of information. For example, the high-order word may contain an x coordinate, and the low-order word a y coordinate. The HIUINT() and LOUINT() utility macros can be used to extract the high- and low-order words of lParam2. The HIUCHAR() and LOUCHAR() utility macros can also be used with HIUINT() and LOUINT() to access any of the bytes. Casting can also be used.

An application may send window messages to any window in the system. If the message is being sent to a window owned by the current thread, then the window proc is called as a subroutine, which is very fast. If the message is being sent to a window of another application or thread, Presentation Manager essentially switches to the appropriate process/thread context and then calls the window procedure. The message is not placed in a queue.

Application threads can control whether or not messages may be received from another thread. Inter-app messages may be received by an thread only in the following circumstances:

* When WinGetMsg, WinPeekMsg, or WinWaitMsg is called
* When sending a message to another app
* When calling a certain set of Presentation Manager routines (such as those that implicitly send messages to other apps)

Note: In order to send messages to a window of another thread, the sending thread MUST have allocated a queue.

5.1.1.1   Mouse and Keyboard Input

Presentation Manager supports user input from both the keyboard or a mouse pointer. A one, two, or three button mouse is supported.

Mouse input is normally directed at the window underneath the mouse cursor. However, an application may direct all mouse input to a particular window (regardless of whether the mouse cursor is in the window) by setting the mouse capture window.

Keyboard input is directed at the keyboard focus. Only one window in the system may be the keyboard focus.

5.1.1.2   Synchronized Input

In Presentation Manager, an input event may not be processed until all previous input has been processed. This is because the destination of an input event, i.e., which application and window the input is intended for, cannot be known until all preceding input has been processed.

For example: A user wants to type a command in one window, use the mouse to activate another window by clicking in it, and then type a command in that window. The routing of the second command to the second window depends on what happens as a result of the processing of the mouse click. Since an application process the mouse click, it may or may not choose to activate the window.

To see how this all works, it's useful to understand how input events are handled and routed by the system:

```
                    +--------------+
                    | Keyboard &   |
                    | Mouse Input  |
                    +--------------+
                           |
                           V
                    +--------------+
                    | System Queue |
                    +--------------+
                           |
                           V
        +--------------------+              +-----------------------+
        |                    |              |                       |
        |   Input Router     |              |  Window Timer,        |
        |   (Keyboard Focus  |              |  Window Repaint,      |
        |    Mouse Capture)   |              |  WinPostMsg() Input   |
        |                    |              |                       |
        +--------------------+              +-----------------------+
           |    |     |                        |     |           |
    +-----------+  +-----------+                |     |           |
    |      +----------) | (------------) | (-------+   |           |
    |      |              +----------) | (-------------+           |
    |      |                        |     |     +-------------------+
    |      |                        |     |     |
    V      V        V     V         V     V
  +-----------+  +-----------+  +-----------+
  | App Queue |  | App Queue |  | App Queue |
  +-----------+  +-----------+  +-----------+
        |              |              |
        V              V              V
  +-----------+  +-----------+  +-----------+
  |   App 1   |  |   App 2   |  |   App 3   |
  +-----------+  +-----------+  +-----------+
```
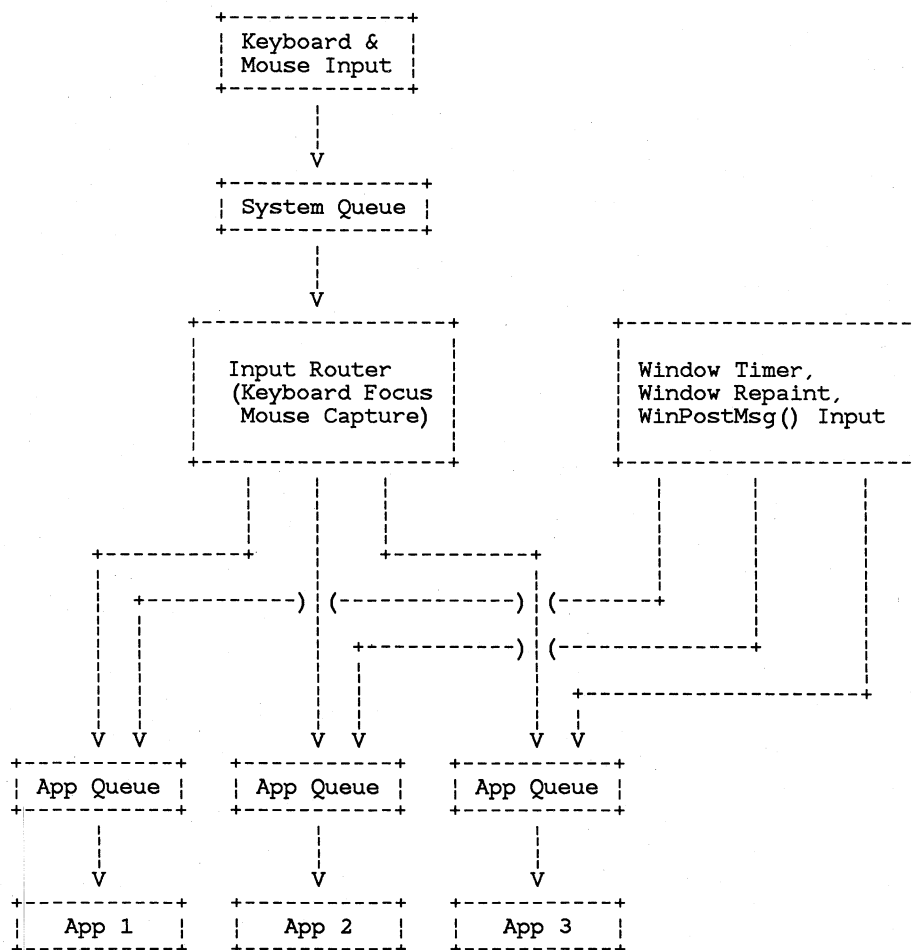
From an application's point of view, all input is read from the application queue with WinGetMsg() or WinPeekMsg() in the form of a QMSG data structure, one message at a time. A QMSG structure contains a window message, which includes the window handle that the input message is intended for.

Before being posted to a particular application queue, all input is first placed in a single queue, called the System Queue. The System Queue is quite large; it's large enough to store 60 or so keypresses and mouse clicks. Only raw, untranslated keyboard input is placed in the system queue.

When an application calls WinGetMsg() or WinPeekMsg(), the next available message in the application queue is read and returned to the application. If there are no messages in the application queue, the system queue is checked for any available input. If an input message is available in the system queue, then the Input Router determines which window and

application the input is destined for. Keyboard messages are posted to the application queue associated with the keyboard focus window. Mouse messages are posted to the queue associated with the mouse capture window or to the window underneath the mouse cursor.

The message is then read out of the queue, and returned to the application.

All of the above processing (except for posting input to the system queue) is performed by WinGetMsg() or WinPeekMsg(). If, during the input routing process, it is determined that the next input event is not intended for the current thread, WinGetMsg() will suspend the current thread, WinPeekMsg() will return FALSE, and the message will be made available to the destination application.

An application may not examine messages in the system queue until the previous message has been processed. A message has not been processed until the application that read the previous message finishes processing the message and calls WinGetMsg() or WinPeekMsg() again for the next item of input.

Notice that in the diagram above, other queued messages such as timers, WM_PAINT messages, and messages directly posted with PostMsg() are not placed in the system queue. These messages are thus not synchronized between applications: for instance, two applications may process a WM_TIMER message at the same time.

5.1.1.3   Input caveats

Mouse button or key down transitions are not placed in the system queue if there is no room for the subsequent up transition. This ensures that applications depending on the state of keys or mouse buttons always accurately reflect the state of the hardware.

WM_MOUSEMOVE messages are handled in a special way: they are not actually queued to prevent overflowing the queue. They are coalesced so that applications receive WM_MOUSEMOVE messages only as fast as they can process them

Autorepeated WM_CHAR keydown messages are coalesced: Consecutive messages are coalesced into a single message, with a repeat count that reflects the number of keydown messages that occured. In this way, applications do not receive these messages faster than they can be processed.

The WinGetMsg() or WinPeekMsg() filter parameters can be used to selectively examine mouse or keyboard input. This also allows keyboard and mouse input to be processed in a different order than it occured.

Messages are not processed until the app processing the previous message calls WinGetMsg() or WinPeekMsg() again.

5.1.2 Message manager functions

5.1.2.1 Data structures

Queue message structure:

```
typedef struct {
    HWND hwnd;
    UINT msg;
    ULONG lParam1;
    ULONG lParam2;
    ULONG time;
    POINT pt;
} QMSG;
```

Message queue handle:

```
typedef HANDLE HMQ;
```

5.1.2.2 Functions

WinSendMsg

Format

```
ULONG WinSendMsg(hwnd, msg, lParam1, lParam2)
HWND   hwnd;
UINT   message;
ULONG  lParam1;
ULONG  lParam2;
```

Description

This function sends a message to a window. The four parameters, hwnd, msg, lParam1, and lParam2 comprise the message being sent. These parameters are passed to the window procedure of the receiving window. The return value is the long value returned by the receiving window's window proc.

Notes

WinSendMsg does not return until the message has been processed. If the window receiving the message belongs to the same thread, the window function is called immediately, as a subroutine. If the window is of another thread or process,

Presentation Manager switches to the appropriate thread and calls the appropriate window function, passing the message to the window function. The message is not placed in the destination thread's queue.

WinTimeoutSendMsg

Format

```
BOOL WinTimoutSendMsg(hwnd, msg, lParam1, lParam2,
        lplResult, dtTimeout)
HWND  hwnd;
UINT  message;
ULONG lParam1;
ULONG lParam2;
ULONG FAR *lplResult;
ULONG dtTimeout;
```

Description

Same as WinSendMsg, except that Win-TimeoutSendMsg is used when sending inter-thread messages. WinTimeoutSendMsg waits up to dtTimeout milliseconds for the receiving thread to reply to the message; if a timeout occurs before the message is replied to, WinTimeoutSendMsg returns FALSE and *lplResult is set to 0L. If no timeout occurs, then WinTimeOutSendMsg returns TRUE, and the message return value is returned in *lplResult.

WinBroadcastMsg

Format

```
BOOL WinBroadcastMsg(hwnd, msg, lParam1, lParam2,
        fSend)
HWND  hwnd;
UINT  message;
ULONG lParam1;
ULONG lParam2;
BOOL  fSend
```

Description

This function sends or posts a message to all top level windows in the system. The hwnd, msg, lParam1, and lParam2 parameters make up the message sent or posted. The message is sent if fSend is TRUE, posted if FALSE. Returns TRUE if all windows were successfully sent or posted to, FALSE otherwise.

See "WinPostMsg"

WinDefWindowProc

Format

```
ULONG WinDefWindowProc(hwnd, msg, lParam1,
        lParam2)
HWND   hwnd;
UINT   message;
ULONG  lParam1;
ULONG  lParam2;
```

Description

This function provides default processing for any window messages that a given application chooses not to process. WinDefWindowProc is generally called in the default case of the window message switch statement, with the parameters passed to the window procedure.

The table below describes the default behavior of messages handled by WinDefWindowProc().

WinInSendMsg

Format

```
BOOL WinInSendMsg(hab)
HAB hab;
```

Description

This function is used to determine whether or not the current thread is processing a message sent by another thread, and if so, whether or not the message was initiated by the activate thread or not. The "active thread" is the thread associated with the current active window.

Returns TRUE if the current thread is processing a message sent by another thread, FALSE otherwise.

See also the WinIsThreadActive function.

Notes

WinInSendMsg typically is used by applications to determine how to proceed with errors when the window that is processing the message is not the active window. For example, if the active window uses WinSendMsg to send a request for information to another window, the other window cannot become active until it returns control from the WinSendMsg call. The only method an inactive

window has to inform the user of an error is to
create a message box.

WinCreateMsgQueue

Format

```
HMQ WinCreateMsgQueue(hab, cMsgs)
HAB hab;
INT cMsgs;
```

Description

Creates a thread queue for a thread with cMsgs
entries. Returns a queue handle if successful,
NULL otherwise. If cMsgs is 0, then the default
queue size is assumed.

Notes   In order to use most Presentation Manager func-
tions, you must call WinCreateMsgQueue.

WinDestroyMsgQueue

Format

```
BOOL WinDestroyMsgQueue(hmq)
HMQ hmq;
```

Description

Destroys the specified thread queue. Returns
TRUE if successful, FALSE otherwise.

Notes   WinDestroyMsgQueue() must be called before kil-
ling a thread or terminating an application.

WinGetMsg

Format

```
BOOL WinGetMsg(hab, lpQmsg, hwndFilter, msgFilterFi
        msgFilterLast);
HAB hab;
LPQMSG lpQmsg;
HWND hwndFilter;
UINT msgFilterFirst, msgFilterLast;
```

Description

WinGetMsg() get a queue message from the thread
queue and returns the message in *lpQmsg.
WinGetMsg() does not return until a message is
available.

WinGetMsg() returns FALSE if a WM_QUIT

message is returned, TRUE otherwise. The return value of WinGetMsg() is generally used to determine when to terminate the application's main loop and exit the program.

Normally, WinGetMsg() is called with hwndFilter == NULL and msgFilterFirst and msgFilterLast == 0. In this case, all messages associated with the current queue are returned, in the order that they were posted. Msgs for windows of other queues are never returned.

If hwndFilter is not NULL, only messages intended for hwndFilter or its children are returned.

If msgFilterFirst and msgFilterLast are non-zero, only messages whose IDs fall in between msgFilterFirst and msgFilterLast are returned. If msgFilterFirst is greater than msgFilterLast, all messages except those that fall between msgFilterLast and msgFilterFirst are returned.

Notes.   By using filtering, messages may be processed in a different order than exist in the queue. Messages that do not match the filter are left in the queue. Filtering is typically used for rearranging the priority of messages by filtering first for higher priority messages followed by lower priority messages, and in situations where it is inconvenient to deal with certain messages. For example, when a mouse down message is received, filtering can be used to wait for the mouse up message without having to worry about recieving other messages.

You must be careful when using filtering with WinGetMsg() to ensure that the message filter will be eventually satisfied so WinGetMsg() will return. For example, calling WinGetMsg() with msgFilterMin and msgFilterMax equal to WM_ CHAR and with hwndFilter set to a window handle that does not have the input focus will prevent WinGetMsg() from returning.

Note that WinGetMsg calls WinTranslateAccel() for keyboard keystrokes. Thus any keystrokes which are used as accelerator keys get translated into WM_ COMMAND or WM_ SYSCOMMAND messages before WinGetMsg returns and are never seen by the application. See the section on Command Key Accelerators for more information.

WinPeekMsg

Format

```
BOOL WinPeekMsg(hab, lpQmsg, hwndFilter,
msgFilterFirst, msgFilterLast, rgf);
HAB hab;
LPQMSG lpQmsg;
HWND hwndFilter;
UINT msgFilterFirst, msgFilterLast;
UINT rgf;
```

Description

WinPeekMsg() is identical to WinGetMsg() except that WinPeekMsg() returns TRUE if a message was returned, FALSE if none were available. The rgf parameter controls whether the returned message is removed from the queue. This parameter is specified with a combination of the following flags ORed together:

WinPeekMsg() Flags

PM_ REMOVE
    Remove message from queue

PM_ NOREMOVE
    Leave message in queue

The message filter parameters (hwndFilter, msgFilterFirst, msgFilterLast) are the same as for WinGetMsg().

WinWaitMsg

Format

```
void WinWaitMsg(hab, msgFilterFirst, msgFilterLast)
UINT msgFilterFirst, msgFilterLast;
```

Description

This function is called to wait until another message fitting the specified filter parameters is posted to the current message queue.

WinQueryMsgPos

Format

```
void WinQueryMsgPos(hab, lppt)
HAB hab;
```

```
LPPOINT lppt;
```

Description

This function returns the mouse position, in screen coordinates, when when the last message obtained from the current message queue was posted. The mouse position is the same as that in the pt field of the QMSG structure.

Notes   To obtain the current position of the mouse cursor instead of the position when the last message occurred, use the WinQueryCursorPos() function.

WinQueryMsgTime

Format

```
ULONG WinQueryMsgTime(hab)
HAB hab;
```

Description

This function returns the message time for the last message retrieved by WinGetMsg() or Win-PeekMsg() from the current message queue. The message time is the time in milliseconds when the message was posted. The time value is the same as that in the time field of the QMSG structure.

Notes   You cannot assume that time values are always increasing. Since the time value is the number of milliseconds since the system was booted, it is possible that the value may wrap around to start again at zero. To accurately calculate time delays between messages, subtract the time of the second message from the time of the first.

WinPostMsg

Format

```
BOOL WinPostMsg(hab, hwnd, msg, lParam1, lParam2)
HWND   hwnd;
UINT   msg;
ULONG  lParam1;
ULONG  lParam2;
HAB    hab;
```

Description

This function posts a message in a window's thread queue. Returns TRUE if successful, FALSE if the queue was full or otherwise unsuccessful.

The four parameters are placed into the queue as part of a QMSG structure. The QMSG time and pt fields are derived from the system time and mouse position at the time WinPostMsg was called.

This function can be used to post messages to any window in the system. The message is posted into the queue associated with the specified window.

If hwnd is NULL, the message is posted into current message queue. When the message is obtained by WinGetMsg() or WinPeekMsg(), the hwnd field will be NULL.

WinPostQueueMsg

Format

```
BOOL WinPostQueueMsg(hmq, msg, lParam1, lParam2)
HMQ   hmq;
UINT  msg;
ULONG lParam1;
ULONG lParam2;
```

Description

This function posts a message to a message queue. Returns TRUE if successful, FALSE if the queue was full or otherwise unsuccessful.

The last three parameters are placed into the queue as part of a QMSG structure. The QMSG hwnd field is set to NULL, and the QMSG time and pt fields are derived from the system time and mouse position at the time WinPostMsg was called.

This function can be used to post messages to any queue in the system.

WinDispatchMsg

Format

```
ULONG WinDispatchMsg(hab, lpQmsg)
HAB hab;
LPQMSG lpQmsg;
```

Description

This function dispatches the message pointed to by lpQmsg on to the window procedure of the window specified by lpQmsg->hwnd. WinDispatchMsg() is

essentially equivalent to:

```
WinSendMsg(lpQmsg->hwnd,
           lpQmsg->msg, lpQmsg->lParam1,
           lpQmsg->lParam2);
```

Notes   The other two fields of the QMSG structure, pt and time, may be obtained by a window procedure with the WinGetMsgPos() and WinGetMsgTime() functions.

WinGetQueueStatus

Format

```
ULONG WinGetQueueStatus(hab)
HAB hab;
```

Description

This function returns a code indicating the status of the message queue associated with the current queue. The hi word of the return value contains bits that indicate what kinds of messages are currently in the queue. The lo word of the return value contains bits that indicate what kinds of messages have been added to the queue since the last call to WinGetQueueStatus().

Each word of the return value contains a combination of the flags shown in the table below.

WinGetQueueState() Flags

QS_ CHAR
    A WM_ CHAR mesage is available.

QS_ MOUSE
    A WM_ MOUSEMOVE or mouse button transition message is available.

QS_ MOUSEMOVE
    A WM_ MOUSEMOVE message is available.

QS_ TIMER
    A WM_ TIMER or WM_ SYSTIMER message is available.

QS_ PAINT
    A WM_ PAINT message is available.

QS_ SEM1
    A WM_ SEM1 message is available.

QS_ SEM2

>A WM_ SEM2 message is available.

QS_ SEM3

>A WM_ SEM3 message is available.

QS_ SEM4

>A WM_ SEM4 message is available.

QS_ POSTMSG

>A posted message other than those listed above is available.

QS_ SENDMSG

>A message has been sent by another application to a window associated with the current queue. In order to receive the message, WinGetMsg() or Win-PeekMsg() should be called.

The low word of the return value (status since last time WinGetQueueMsg() called) will not contain QS_ bits that do not also exist in the hi word. In other words, if there are no messages in the queue, the low word of the return value is 0.

Note   This function is very fast. It is typically used inside loops to determine whether WinGetMsg() or WinPeekMsg() should be called to process input.

WinSetMsgInterest

Format

```
VOID WinSetMsgInterest (hWnd, msg_class, control)
HWND   hWnd;
UINT   msg_class;
INT    control;
```

Description

WinSetMsgInterest is used by an application to indicate the messages that the window hwnd wants to process. It also indicates that any other messages sent to the window will be processed by the application in a default manner. This means that the application will route *all*

msg_ class specifies either a single message ID (e.g. WM_ SHOW) or the ID of a *Group* of messages. A message group is a logical combination of a number of message IDs which can be used to facilitate bulk enablement/disablement of messages. Group IDs are as follows:

Message Group IDs.

SMI_ ALL

> All messages are included in this group.

TBD   many other groupings of messages are
       yet to be identified.

The Control parameter indicates whether the
application is interested or not interested in receiv-
ing the specified messages, by setting the following
values:

SMI_ INTEREST

> indicates interest in the messages.

SMI_ NOINTEREST

> indicates not interested in the messages.

The default state after window creation is deter-
mined by the options setting used for the WinIni-
tialise call. For Presentation Manager, the default
is as if interest in all messages had been notified to
the system. Thus all messages will be sent to the
application for either processing or passing to the
default window procedure.

Note that in fact, Presentation Manager treats this
call as a NO-OP. All messages are always given to
the application. However, other systems imple-
menting this function will make use of this func-
tion to offer potential performance gains and so
portable applications should use this function.

Each use of the function is then incremental - i.e.
the call adds to or subtracts from the previous set
of messages.

For example, the call

```
WinSetMsgInterest(hwnd, SMI_ALL, SMI_INTEREST)
```

gives the application total control of all the mes-
sages which the system generates.

Indicating which messages the application intends
to process itself offers the system the opportunity
to optimise processing of those messages which the
application is not going to process directly.

Note however, that if an application is to be
ported between different systems, it must not
depend upon the receipt of messages not supported
by one of the systems. Conversely, the application

should expect that any system has the *potential* to generate messages not present on other systems. It is important that the application pass on any such messages to the default window procedure. It is guaranteed that the default processing of such messages will be acceptable in terms of User Interface Standards on all systems.

5.1.2.3 WinDefWindowProc default behaviors

Below is a list of the messages processed by WinDefWindowProc() and the default actions taken.

Msg ID   Action taken by WinDefWindowProc()

WM_ SETWINDOWPARAMS
> Sets caption text, if text is specified.

WM_ GETWINDOWPARAMS
> Gets caption text, if text is specified.

WM_ CANCELMODE
> Cancels any internal mode loop (scrolling, size/move tracking, etc)

WM_ PAINT
> Calls WinBeginPaint/EndPaint

WM_ QUERYENDSESSION
> Returns TRUE

WM_ SYSCOMMAND
> Standard processing of system commands (size/move tracking, etc)

WM_ ACTIVATE
> If activating, sets focus to hwnd.

WM_ SHOWWINDOW

WM_ DOSUSPEND

5.1.2.4 Keyboard Input messages

5.1.2.4.1 Keyboard input

Keyboard input is obtained in the form of messages received via WinGetMsg.

A WM_CHAR message is delivered for each keydown and keyup for all keys on the keyboard. Translation of the scan code received from the keyboard is done by the WinGetMsg call that receives the keystroke.

WinGetMsg does not send a message for every typamatic repeat from the keyboard; it may buffer typamatic repeats into one or more messages. nt Each message contains a count, which is the number of typamatic repeats that have occurred since the first keydown, or since the last wingetmsg call. This count will begin at one for the first keydown.

Keyboard data along with mouse data is buffered asynchronously into the Presentation Manager system queue. Keyboard data is removed from the system queue w hen the application that owns the input focus calls WinGetMsg or WinPeekMsg. Only one keyboard event is dequeued at a time.

Translation occurs when the event is dequeued. The message obtained from WinGetMsg contains three separate fields that represent the key pressed: the hardware dependent scan code, the virtual key code and the codepoint or dead key. These are discussed below:

- The virtual key (VKEY) concept is that there should be a virtual key code for each "word" (eg esc or F1) on the keytops of the keyboard. Consistency requires that most applications should use the PC set of virtual keys built in to Presentation Manager, but applications with special requirements can define their own virtual key sets. An example of valid use of this capability is mainframe applications accessed via a terminal emulator. These use words such as clear, PA1, etc in contrast to PC applications which use esc, home etc.

- The code point (CKEY) concept is that there should be a code point value for every key on the keyboard with a symbol on it. The code points can be either ASCII or EBCDIC and are country dependent. Effectively, each code point corresponds to a unique "glyph" that can appear on the screen.

- Some keys with words on them (eg Enter) generate both a virtual key value and a code point. This is because the key does need to be treated as a function key in some applications, but also has a defined ASCII code point associated with it which some applications may prefer to use.

- For CKEY values that correspond to dead keys (e.g. umlaut) WinGetMsg will identify these CKEYs with a special flag in the WM_CHAR message. It is the application's responsibility to echo the dead key in the appropriate manner (i.e. without advancing the

cursor). If the next CKEY after the dead key is a valid dead key combination, then another flag will be set in the WM_CHAR message to identify the composite character. Again it is the application's responsibility to echo the character appropriately. There are three situations the application must deal with:

— valid dead key combination should replace the dead key display with the new composite character

— invalid dead key combination (except the space character) should leave the dead key displayed, advance the cursor and display the new CKEY, followed by a beep.

— dead key followed by a space should leave the dead key displayed and advance the cursor.

● The valid set of dead keys and their combinations with other keys is defined for each supported code page.

5.1.2.4.2 Keystroke Translation

Presentation Manager keystroke translation provides full flexibility in remapping, support of country specific keyboard layouts, and support of EBCDIC and ASCII code pages. This is achieved by the use of three types of table which are described below:

● The key to VKEY table (VKeyXLateTbl). This table generates virtual key codes based on the key pressed and the shift state. Presentation Manager supplies a two tables of this type covering the PC VKEY set for the two physically different keyboards. These tables cover all languages and keyboard layouts.

● The key to Universal Glyph List (UGL) table(GlyphXLateTbl). (The UGL is a list of all (non-DBCS) glyphs that can be generated by a Presentation Manager application using standard Presentation Manager facilities. All glyphs for all supported languages, plus the APL glyphs, are included in the UGL.) The key to UGL table is always used in conjunction with the UGL to CKEY table described below. It is uniquely defined by the layout of the keytops. Presentation Manager supplies tables of this type corresponding to all supported keyboards.

● - The UGL to CKEY table (CharXLateTbl). This table is used in conjunction with the previous one. Presentation Managersupplies a table of this type for each supported code page. Also included in this table is the dead key table, which defines the valid dead keys for each code page and the valid dead key combinations.

5.1.2.4.3 API Calls

The API calls allow the application to control the translation process that generates the virtual key and character code values that are in the WM_ CHAR message. The translation process consists of the following steps:

- Apply scan code/keyboard state to VKeyXLateTbl. Result is a virtual key.

- Apply scan code/keyboard state to GlyphXLateTbl. Result is a glyph code.

- Apply the glyph code to CharXLateTbl. Result is a character code, with appropriate dead key bits set.

The Presentation Manager keyboard driver is a dynlink library, that contains all of   the system provided versions of the above tables as resource segments. There are three API calls for accessing these tables, as well as translation tables defined as resources in the application's executable image:

```
WinLoadVKeyXLateTbl( idModule, keyboard ) => hXLate
WinLoadGlyphXLateTbl( idModule, keyboard, country ) => hXLate
WinLoadCharXLateTbl( idModule, codepage ) => hXLate
```

where all three return a handle to a translation table, whose format is defined below. The idModule parameter, returned by the DOS DosLoad-Module call, is either -1 for accessing the tales in the Presentation Manager keyboard driver, 0 for accessing tables in the application's executable image or a module handle of some other dynlink module.

In order to allow an application to dynamically create a translation table in memory at runtime, another API call is provided that takes a far pointer to a translation table in memory and returns a translation table handle that can be used in the remaining API calls.

```
WinCreateXLateTbl( hab, lpXLateTbl ) => hXLate
```

along with the following API call to destroy the handle created with Win-CreateXLateTbl:

```
WinDestroyXLateTbl( hXLate )
```

Associated with each message queue are handles for the three translation tables needed by the translation process. The following API calls allow an application to override any or all of these translation tables.

```
WinSetVKeyXLateTbl( hab, hXLate ) => hOldVKeyXLate
WinSetGlyphXLateTbl( hab, hXLate ) => hOldGlyphXLate
WinSetCharXLateTbl( hab, hXLate ) => hOldCharXLate
```

The WinGetMsg function will use these three translation table handles to call the KeyTranslate function, which is a private API exported by the Presentation Manager keyboard driver for use by the WinGetMsg function. This API looks like:

```
KeyTranslate( lpMsg, scan, fBreak, hVKeyXLate,
              hGlyphXLate, hCharXLate )
```

and will fill in the passed message structure with the appropriate information. Internally the KeyTranslate function will maintain the current up/down/toggle state of each scan code and any pending dead key. This function assumes that it will see ALL key transitions.

5.1.2.4.4 PC Virtual Keys

Presentation Manager supplies as part of the toolkit INCLUDE files defining the PC VKEY set.

The PC VKEY set is shown in the table below.

— VK_F1	
— VK_F2	
— VK_F3	
— VK_F4	
— VK_F5	
— VK_F6	
— VK_F7	
— VK_F8	
— VK_F9	
— VK_F10	
— VK_F11	(∗)
— VK_F12	(∗)
— VK_F13	(∗)
— VK_F14	(∗)
— VK_F15	(∗)
— VK_F16	(∗)
— VK_F17	(∗)
— VK_F18	(∗)

— VK_F19   (∗)
— VK_F20   (∗)
— VK_F21   (∗)
— VK_F22   (∗)
— VK_F23   (∗)
— VK_F24   (∗)

—

— VK_PA1   (∗)
— VK_PA2   (∗)
— VK_PA3   (∗)

—

— VK_SHIFT   (+)
— VK_CONTROL   (+)
— VK_ALT   (+)
— VK_ALTGRAF   (+)   (∗)

—

— VK_BACKSPACE   (+)
— VK_TAB   (+)
— VK_BACKTAB   (+)
— VK_ENTER
— VK_CLEAR   (∗)
— VK_CAPSLOCK   (+)
— VK_NUMLOCK   (+)
— VK_SCRLOCK   (+)
— VK_ESCAPE   (+)
— VK_UP   (+)
— VK_DOWN   (+)
— VK_LEFT   (+)
— VK_RIGHT   (+)
— VK_HOME   (+)
— VK_END   (+)
— VK_PAGEUP   (+)

- VK_PAGEDOWN   (+)
- VK_INSERT   (+)
- VK_DELETE   (+)
- VK_PRINTSCREEN   (+)
- VK_COPYTOPRINTER (+)
- VK_BREAK   (+)
- VK_SYSREQ   (*)
- VK_HELP   (*)
- VK_SELECTORLIGHTPENATTN (*)

-

- VK_BUTTON1   (=)
- VK_BUTTON2   (=)
- VK_BUTTON3   (=)

Note that the keys marked with a * will not be generated by all keyboards. Applications using them should provide an alternative.

Note that keys marked with = are never generated with WM_CHAR messages, but can be used with WinQueryKeyState() to query the state of the mouse buttons.

WM_CHAR

Format

```
WM_CHAR
LOUCHAR(LOUINT(lParam1)): UCHAR brgf;
HIUCHAR(LOUINT(lParam1)): UCHAR scancode;
HIUINT(lParam1):          UINT cRepeat;
LOUINT(lParam2):          UINT ch;
HIUINT(lParam2):          UINT vk;
Returns:                  BOOL fProcessed;
```

Description
The WM_CHAR messages is posted as the result of keyboard events. ch contains the character value (translated according to the current codepage) resulting from the keyboard event. vk contains the virtual key code, if there is one, otherwise 0. cRepeat contains the repeat count. scancode contains the hardware scancode for the key that was pressed.

The low order byte of lParam1 contains a combination of the values shown below. The KC_ constants are defined as ULONG constants so they may be ANDed directly with lParam1 to test the flags in brgf.

Code   Meaning

KC_CHAR
>   The character value is valid. (Otherwise, the character field contains 0.

KC_SCANCODE
>   The scan code is valid. Otherwise, the scan code field contains 0. Generally, all WM_CHAR messages generated from actual user input have KC_SCANCODE set. However, if the message was generated by an application that has issued the WinSetHook() function to filter keystrokes, or has been posted to the application queue, this bit may not be set.

KC_VIRTUALKEY
>   The virtual key value is valid. Otherwise, the virtual key field contains 0.

KC_KEYUP
>   The event was a key up transition. (Otherwise, it was a down transition).

KC_PREVDOWN
>   The virtual key was previously down. (Otherwise, it was previously up.)

KC_DEADKEY
>   means that the char. code is a dead key. Application responsible for displaying the glyph for the dead key without advancing the cursor.

KC_COMPOSITE
>   means that the char. code was formed by combining the current key with the previous dead key.

KC_INVALIDCOMP
>   means that the char. code was not a valid combination with the preceeding dead key. Application responsible for advancing the cursor past the dead key glyph and then if the current character is NOT a space, beeping the speaker and displaying the new char. code.

The application window procedure should return TRUE if it processes a particular WM_CHAR message, FALSE otherwise.

Notes   Virtual keys are not defined for keys 'A' - 'Z', 'a' - 'z', '0' - '9', or "special" keys. They are only defined for basic keys that will probably be country dependent.

5.1.2.5 Keyboard functions

WinGetFocus

Format

```
HWND WinGetFocus(hab, fLock)
HAB hab;
BOOL fLock;
```

Description

This function returns the focus window, or NULL
if there is no focus window. If fLock is TRUE, the
window is returned locked, otherwise it is returned
unlocked.

WinSetFocus

Format

```
HWND WinSetFocus(hab, hwnd)
HWND hwnd;
HAB hab;
```

Description

This function sets the input focus to hwnd.
Returns the handle of the window that previously
had the focus, or NULL if no window had the
focus. If hwnd is NULL, then no window has the
input focus. Returns an unlocked window handle.
When WinSetFocus() is called, the following events
take place:

1. A WM_SETFOCUS message with fFocus ==
   FALSE is sent to the current focus window,
   unless it is NULL.

2. A WM_ACTIVATE message is sent with fAc-
   tive == FALSE to the current active window,
   unless it is NULL, or the active window is not
   changing.

3. If a new application is being made the active
   application, then a WM_ACTIVATETHREAD
   message with fActive == FALSE is sent to the
   the current active application.

4. The new focus window, active window, and
   active application are established.

5. If a new application is being made the active application, then a WM_ACTIVATETHREAD message with fActive == TRUE is sent to the the new active application.

6. The top level window associated with the new focus window is sent a WM_ACTIVATE with fActive == TRUE, and fSetFocus = FALSE, but only if this window is different than the current active window.

7. The new focus window is sent a WM_SETFOCUS message.

Notes.   During the processing of a WinSetFocus() call, if WinGetActiveWindow or WinGetFocus() are called, the old active and focus windows are returned until the new ones have been established. In other words, even though WM_SETFOCUS (false) and WM_ACTIVATE(false) messages may have been sent to the old windows, those old windows are considered to be active and have the focus (until the system establishes the new active and focus windows).

If WinSetFocus() is called during WM_ACTIVATE or WM_ACTIVATETHREAD message processing, a WM_SETFOCUS message with fFocus == FALSE is NOT sent, since no window has the focus.

WM_SETFOCUS

Format

```
WM_SETFOCUS
lParam1: HWND hwndFocus
lParam2: BOOL fFocus
```

Description
This message is sent by SetFocus() to the window receiving or losing the input focus. hwndFocus is the window that previously had the input focus, which is NULL if no window previously had the focus. If fFocus==FALSE, the window is losing the focus, and hwndFocus is the window handle of the window that will be receiving the input focus (which may be NULL).

The WM_SETFOCUS message is sent before SetFocus() returns.

Except in the case of the WM_ACTIVATE

message with fActive == TRUE, an application processing WM_SETFOCUS, WM_ACTIVATE, or WM_ACTIVATETHREAD should not change the focus window or active window. If it does, focus and active window must be restored before the application returns from processing the message. For this reason, any dialog boxes or windows brought up during WM_SETFOCUS, WM_ACTIVATE or WM_ACTIVATETHREAD processing should be system modal.

WinGetKeyState
_____

Format

```
INT WinGetKeyState(hab, vk)
HAB hab;
INT vk;
```

Description

The WinGetKeyState() function is used to determine whether a virtual key is up, down, or toggled. vk may be one of the VK_ values shown in the table above.

The 0x8000 bit is set (less than 0) if the key is down, otherwise the key is up. If the 0x0001 bit is set, then the key is toggled. A key is toggled if it has been pressed an odd number of times since the system was started.

Notes   This function returns the state of the key at the time that the last message obtained from the queue was posted. See GetPhysKeyState().

WinGetKeyState() can be used to obtain the state of the mouse buttons with the VK_BUTTON1, VK_BUTTON2, and VK_BUTTON3 virtual key codes.

5.1.2.6   Mouse Input

There are two forms of mouse input: mouse button transitions and mouse pointer movement. Three mouse buttons are supported. These are known as mouse buttons 1, 2 and 3. Only button 1 is supported for a single-button mouse, buttons 1 and 2 for a two-button mouse. Button double clicks are supported as well: if a mouse button goes down within a certain amount of time and within a certain screen area from the last time the button went down, a doubleclick down message is posted.

The mouse buttons have virtual key codes assigned to them, though a WM_CHAR message is not posted when a transition occurs. These codes are shown in the table below. As with other virtual keys, it is possible to poll the state of any of the mouse buttons with the WinGetKeyState() function, using the mouse button virtual keycodes.

The WM_MOUSEFIRST and WM_MOUSELAST constant values can be used with WinGetMsg() and WinPeekMsg() to filter for all mouse input only. These values do not imply the separate existence of messages WM_MOUSEFIRST and WM_MOUSELAST!

5.1.2.6.1 *Mouse Usage and Mouse Button Assignments*

Presentation Manager recommends certain standardised usage of mouse buttons and also provides support for mouse button assignment for left and right handed users, as follows:

1. All programs should support use of a mouse.

2. Button assignments for mouse support are:

   1. Button 1: Select button

   2. Button 2 (if available): Application defined. If scroll lock is implemented and button 2 is available, scroll lock should be assigned to button 2.

   3. Button 3 (if available): Reserved for system use

3. On a two button device, simultaneous pressing of both emulates button three

4. The user must be allowed the option of using a right- or left-handed mouse.

   1. Right-handed: Buttons numbered left-to-right as illustrated in the following figure:

```
+--------------------+   +--------------------+   +---------------------+
| +---+  +---+  +---+ |   | +---+   +---+      |   | +---+     +---+      |
| |   |  |   |  |   | |   | |   |   |   |      |   | |   |     |   |      |
| | 1 |  | 2 |  | 3 | |   | | 1 |   | 2 |      |-  | | 1 |     | 2 |      |
| |   |  |   |  |   | |   | |   |   |   |      |   | |   |     |   |      |
| +---]  +---]  +---] |   | +---]   +---]      |   | +---]     +---]      |
|                     |   |                    |   |                     |
|                     |   |                    |   |                     |
|                     |   |                    |   |                     |
|                     |   |                    |   |                     |
|                     |   |                    |   |                     |
|                     |   |                    |   |                     |
|                     |   |                    |   |                     |
+--------------------]    +-------------------]    +--------------------]
      3-Button                  2-Button                   2-Button
```
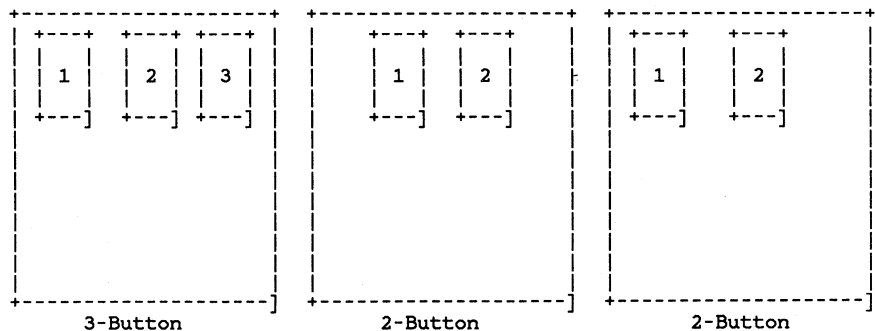
Figure 5.1 Right-handed Button Arrangement

2. Left-handed: Buttons numbered right-to-left as illustrated in
    the following figure:

```
+--------------------+  +--------------------+  +--------------------+
| +---+ +---+ +---+  |  |     +---+ +---+    |  | +---+     +---+    |
| |   | |   | |   |  |  |     |   | |   |    |  | |   |     |   |    |
| | 3 | | 2 | | 1 |  |  |     | 2 | | 1 |    |  | | 2 |     | 1 |    |
| |   | |   | |   |  |  |     |   | |   |    |  | |   |     |   |    |
| +---] +---] +---] |  |     +---] +---]    |  | +---]     +---]    |
|                   |  |                    |  |                    |
|                   |  |                    |  |                    |
|                   |  |                    |  |                    |
|                   |  |                    |  |                    |
|                   |  |                    |  |                    |
|                   |  |                    |  |                    |
+-------------------]  +--------------------]  +--------------------]
     3-Button               2-Button               2-Button
```
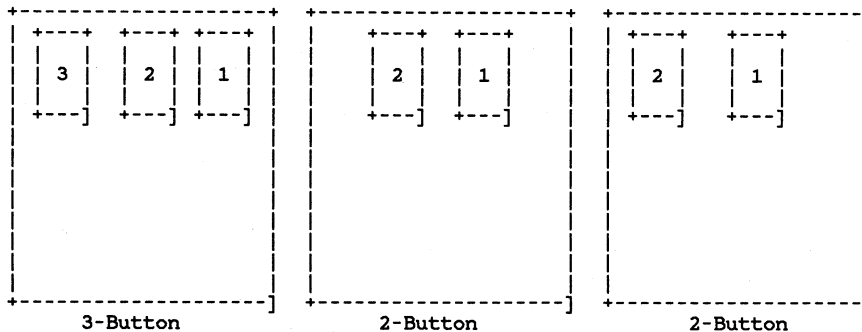
Figure 5.2 Left-handed Button Arrangement

5. The following button techniques are supported:

    1. "Press and release" on the same item or in the same space (e.g.
        you can press and release in "white space"). An intervening
        move off the item then back on while the button is held pressed
        is OK. If the button is then released it is treated the same as
        pressing and releasing while staying on the item.

        Press and release is also called "clicking".

    2. "Press and release twice", which is accomplished by performing
        a press and release within a prescribed (short) time period.

        Press and release twice is also called "double clicking".

    3. "Press and hold," where an action continues while the button is
        held and ceases when the button is released.

6. On one button devices the functions available from buttons 2 and
    3 are performed from the keyboard using their assigned keys.

5.1.2.7 Mouse Capture functions

Normally an application only gets mouse input when the mouse cursor is
over one of its windows. The mouse capture functions allow an application
to track the mouse and get all the mouse input no matter where the mouse
cursor goes. Applications for this are usually for dragging 'objects' on the
screen, capturing the mouse once the mouse button goes down, releasing
once the button goes up.

WinSetCapture

Format

```
HWND WinSetCapture(hab, hwnd)
HWND hwnd;
HAB hab;
```

Description

WinSetCapture() assigns the mouse capture to hwnd. If hwnd is NULL, then the mouse capture is released. Returns the window handle of the window that previously captured the mouse, or NULL if no capture was set.

With the mouse capture set to a window, all mouse input is directed to that window, regardless of whether the mouse is over that window.

Notes

When WinSetCapture(hab, NULL) is called to release the mouse capture, a WM_MOUSEMOVE message is posted regardless of whether the mouse pointer has actually moved. This is to ensure that the window underneath the mouse at that time has a chance to change the mouse cursor shape, etc.

WinSetCapture() returns an unlocked window handle.

WM_CANCELMODE

Format

```
WM_CANCELMODE
lParam1:
lParam2:
Returns:
```

Description

This message is sent when a dialog box or message box is posted to the window with the mouse capture to terminate any modeloops that may be in effect.

WinGetCapture

Format

```
HWND WinGetCapture(hab, fLock)
HAB hab;
BOOL fLock;
```

This function returns the window handle that has the mouse capture. If fLock is TRUE, the window handle returned is locked, and WinUnlockWindow(hwnd) must be called at some point. If fLock is FALSE, the window handle returned is unlocked.

WM_ MOUSEMOVE

Format

```
WM_MOUSEMOVE
LOUINT(lParam1):  INT   xMouse;
HIUINT(lParam1):  INT   yMouse;
lParam2:          UINT  wHitTest;
Returns:          BOOL  fProcessed;
```

Description

The WM_ MOUSEMOVE message is posted when the mouse pointer moves. lParam1 contains the position of the mouse in window coordinates, relative to the bottom left corner of the window: LOUINT(lParam1) has the X coordinate, and HIUINT(lParam1) has the Y coordinate. LOUINT(lParam2) contains the result of the WM_ HITTEST message, or 0 if a mouse capture is currently in progress. The window proc should return TRUE if it processes the message, FALSE otherwise.

WM_ BUTTON1DOWN

Format

```
WM_BUTTON1DOWN
LOUINT(lParam1):  INT   xMouse;
HIUINT(lParam1):  INT   yMouse;
lParam2:          UINT  wHitTest;
Returns:          BOOL  fProcessed;
```

WM_ BUTTON1UP

Format

```
WM_BUTTON1UP
LOUINT(lParam1):  INT   xMouse;
HIUINT(lParam1):  INT   yMouse;
lParam2:          UINT  wHitTest;
Returns:          BOOL  fProcessed;
```

WM_BUTTON1DBLCLK

Format

```
WM_BUTTOND1BLCLK
LOUINT(lParam1):  INT   xMouse;
HIUINT(lParam1):  INT   yMouse;
lParam2:          UINT  wHitTest;
Returns:          BOOL  fProcessed;
```

WM_BUTTON2DOWN

Format

```
WM_BUTTON2DOWN
LOUINT(lParam1):  INT   xMouse;
HIUINT(lParam1):  INT   yMouse;
lParam2:          UINT  wHitTest;
Returns:          BOOL  fProcessed;
```

WM_BUTTON1UP

Format

```
WM_BUTTON1UP
LOUINT(lParam1):  INT   xMouse;
HIUINT(lParam1):  INT   yMouse;
lParam2:          UINT  wHitTest;
Returns:          BOOL  fProcessed;
```

WM_BUTTON1DBLCLK

Format

```
WM_BUTTON1DBLCLK
LOUINT(lParam1):  INT   xMouse;
HIUINT(lParam1):  INT   yMouse;
lParam2:          UINT  wHitTest;
Returns:          BOOL  fProcessed;
```

WM_BUTTON3DOWN

Format

```
WM_BUTTON3DOWN
LOUINT(lParam1):  INT   xMouse;
HIUINT(lParam1):  INT   yMouse;
lParam2:          UINT  wHitTest;
```

Returns: BOOL fProcessed;

WM_ BUTTON1UP

Format

```
WM_BUTTON1UP
LOUINT(lParam1):  INT  xMouse;
HIUINT(lParam1):  INT  yMouse;
lParam2:          UINT wHitTest;
Returns:          BOOL fProcessed;
```

WM_ BUTTON1DBLCLK

Format

```
WM_BUTTON1DBLCLK
LOUINT(lParam1):  INT  xMouse;
HIUINT(lParam1):  INT  yMouse;
lParam2:          UINT wHitTest;
Returns:          BOOL fProcessed;
```

Description

The mouse transition messages above are posted when either of the three mouse buttons go up or down. lParam1 contains the position of the mouse in window coordinates, relative to the top left corner of the window: LOUINT(lParam1) has the X coordinate, and HIUINT(lParam1) has the Y coordinate. LOUINT(lParam2) contains the result of the WM_ HITTEST message, or 0 if a mouse capture is currently in progress. The window proc should return TRUE if it processes the message, FALSE otherwise.

Normally, WM_ BUTTON?UP and WM_ BUTTON?DOWN messages are posted when mouse buttons go up or down. However, a WM_ BUTTON?DBLCLK message will be posted in in place of a WM_ BUTTON?DOWN if the following conditions are met:

1. The button down occured within a certain amount of time since the last button down (typically 1/4 second).

2. Both button downs occured in the same window, and within a certain distance from each other (typically an area roughly the size of two characters).

WM_HITTEST

Format

```
WM_HITTEST
lParam1:        POINT ptMouse;
lParam2:        OL
Returns:        UINT wHitTest;
```

Description

This message is sent to a window by WinGet/PeekMsg() when determining whether or not the message is in fact destined for the window. The application may return one of the following values:

Value. Meaning

0: The message should be processed as normal: i.e., a WM_BUTTON* or WM_MOUSEMOVE message should be posted to the window.

HT_TRANSPARENT:
The part of the window underneath the mouse cursor is "transparent"; hit-testing should continue on windows underneath this window, as if this window did not exist.

HT_DISCARD:
The message should be discarded: no message should be posted to the application.

HT_ERROR:
Like HT_DISCARD, except that if the message is a button down message, an alarm will sound.

WinDefWindowProc() handles this message by returning HT_ERROR if the window is disabled, and 0 otherwise.

Notice that it is the handling of this message which determines whether or not a disabled window may process mouse clicks.

ptMouse is in window coordinates.

5.1.2.8 Mouse Tracking functions.

Functions are provided for tracking mouse movements with a rectangle on the screen, as follows.

WinTrackRect

Format

```
BOOL WinTrackRect(hwnd, hps, lpti)
HWND hwnd;
HPS hps;
TRACKINFO lpti;
```

Description

This is a general purpose mouse tracking routine. WinTrackRect() draws a rectangle at a specified location and allows the user to position the entire rectangle or size a specific side or corner smaller or larger. The resultant rectangle is then returned to the application, which can use this new information for size and position data. For example, the window manager interface for moving and sizing windows via the wide sizing borders simply calls WinTrackRect().

WinTrackRect() allows the caller to control such limiting values as:

- A maximum and minimum tracking size

- An absolute tracking position limits

- The tracking rectangle side widths

- A restriction of tracking rectangle movements to a pre-defined positional grid.

WinTrackRect() is called with a long pointer that points to a TRACKINFO structure:

```
typedef struct tagTRACKINFO {
    int cx;
    int cy;
    int cxGrid;
    int cyGrid;
    RECT rcTrack;
    RECT rcBoundary;
    POINT ptMinTrack;
    POINT ptMaxTrack;
    WORD rgf;
} TRACKINFO;
```

If the passed hps is NULL, hwnd is used to calculate an hps for tracking (hwnd is assumed to be the

window the tracking is taking place in). lpti is a long pointer to a structure called TRACKINFO. In this structure, rcTrack is the start rectangle. It is modified as the rectangle is tracked, holding the new tracking position on exit.

cx is the width of the left and right tracking sides, cy is the height of the top and bottom tracking sides. cxGrid and cyGrid define a positional grid that all tracking movements will be bound to. rcBoundary is an absolute bounding rectangle that the tracking rectangle cannot extend completely out of (there are two kinds of boundary detection with rcBoundary, defined by the flag TF_LIMITBOUNDARY).

ptMinTrack defines the minimum x and y tracking sizes , and ptMaxTrack defines the maximum tracking sizes. rgf is a bit array of tracking flags specifying what tracking operation should take place. TRUE is returned if tracking was success-ful, FALSE returned if tracking is canceled, or if the mouse was already captured when Win-TrackRect() is called. Only one tracking rectangle may be in use at one time.

If the passed hps is NULL, then the hps will calcu-lated with the assumption that the window is not a WS_CLIPCHILDREN window. In other words, when the drag rectangle appears, it will not be clipped by any children within the window. If a window is a WS_CLIPCHILDREN window, and the application wants the drag rectangle to be clipped, it must explicitly pass an hps.

The tracking flags may be or'ed together. They are:

TF_LEFT
> Track the left side of the rectangle.

TF_TOP
> Track the top side of the rectangle.

TF_RIGHT
> Track the right side of the rectangle.

TF_BOTTOM
> Track the bottom side of the rectangle.

TF_MOVE
> Track all sides of the rectangle.

TF_ KEYBOARD
>   Tracking starts with keyboard interface.

TF_ GRID
>   Restrict tracking to a grid defined by
>   cxGrid and cyGrid.

TF_ STANDARD
>   cx, cy, cxGrid, cyGrid are all multiples of
>   cxBorder and cyBorder.

TF_ LIMITBOUNDARY
>   Make sure the tracking never extends
>   past rcBoundry. The defualt behavior is
>   to make sure a minimum part of the
>   tracking tracking rectangle is always
>   within rcBoundary. This minimum size is
>   defined by cx and cy.

If the TF_ KEYBOARD flag is included, the mouse
pointer is positioned to the center of the tracking
rectangle. Otherwise the mouse pointer is not
moved from its current position. At this point
there is an established delta between the mouse
position and the part of the tracking rect it moves,
and this value is kept constant.

While moving or sizing with the keyboard inter-
face, the mouse pointer is repositioned along with
the tracking rectangle's new size or position.

While tracking, these keys are active:

ENTER   Accepts the new position or size.

LEFT   Moves the mouse pointer and tracking
        rectangle left.

UP   Moves the mouse pointer and tracking
        rectangle up.

RIGHT   Moves the mouse pointer and tracking
        rectangle right.

DOWN   Moves the mouse pointer and tracking
        rectangle down.

ESCAPE
>   Cancels the current tracking operation.

The mouse and the keyboard interface are inter-
mixable. The caller doesn't have to include the
TF_ KEYBOARD flag to be able to use the key-
board interface; this simply initializes the position
of the mouse pointer.

The tracking rectangle is usually logically "on top" of objects it track s, so that the user can see the old size and position while tracking the new. Because of this, it is conceivable that a window "below" the tracking rectangle can update while part of the tracking rectangle is "above" it.

Since the tracking rectangle is drawn in exclusive-or mode, no window may draw below the tracking rectangle (and thereby obliterate it) without first notifying the tracking code, because undesirable chunks of tracking rectangle may be left behind. If the window doing the drawing is clipped out from the window the tracking is occuring in, there is no problem.

To catch the general case where a window processing a WM_PAINT message might draw over the tracking rectangle, Windows treats the tracking rectangle as a system wide resource. Only one may be in use at any one time. If the currently updating window has a chance of drawing on the tracking rectangle, Windows will remove the tracking rectangle while that window and its children update, and then replace it. This is specifically done inside of WinBeginPaint() / WinEndPaint(). If the tracking rectangle overlaps, it will be removed in WinBeginPaint(). In WinEndPaint() all children will be updated via WinUpdateWindow() before the tracking rectangle is redrawn.

Notes   WinTrackRect() has a modal loop within its function. The loop has a WH_MSGFILTER hook and hook code, MSGF_TRACK. Refer to the Hook documentation for an explanation of this hook type.

There are several cases that windows update their images as a result of some message other than a WM_PAINT. For this reason, an interface has be en provided for application use, to preserve the integrity of the tracking rectangle image:

WinShowTrackRect

Format

```
INT WinShowTrackRect(hwnd, fShow)
HWND hwnd;
BOOL fShow;
```

Description

hwnd is the window handle passed in to Win-TrackRect(), the window the tracking is taking place in. WinShowTrackRect() manages a show count. When a hide request is made (fShow is FALSE), this count is decremented. When a show request is made (fShow is TRUE), this count is incremented. When the count makes a transition from 0 to -1, the rectangle is hidden. When the count makes a transition from -1 to 0, the rectangle is shown.

If rectangle tracking, the application should call this routine to hide the rectangle if there's possibility of corrupting the track rectangle while drawing, showing it afterward. Since rcTrack in the TRACKINFO structure is updating continuously, the application can examine the current tracking rectangle coordinates to determine whether temporary hiding is necessary .

The only case where an application needs to call WinShowTrackRect is in the case of asynchronous drawing. If an application is drawing on one thread, and issuing WinTrackRect on another, undesirable pieces of tracking rectangle may be left behind. The drawing thread is therefore responsible for issuing WinShowTrackRect when tracking may be in progress. The application should provide for communication between the two threads to ensure that if the one thread is tracking the drawing thread will issue WinShowTrackRect. This could be done using a semaphore, for example.

5.1.2.9 WM_ SEMN MESSAGES.

WM_ SEMn messages are designed to facilitate the operation of applications that have other non Presentation Manager threads which:

1. Require to wait on external non Presentation Manager events

2. Require to signal the main Presentation Manager thread in the app

There are four semaphore messages.

```
WM_SEM1
lParaml:      ULONG lrgfAccumBits
lParam2:      OL
Returns:      OL
```

```
WM_SEM2
lParaml:      ULONG lrgfAccumBits
lParam2:      OL
Returns:      OL

WM_SEM3
lParaml:      ULONG lrgfAccumBits
lParam2:      OL
Returns:      OL

WM_SEM4
lParaml:      ULONG lrgfAccumBits
lParam2:      OL
Returns:      OL
```

The messages shown above are a special set of 4 messages that are similar to standard messages except that if more than one is posted before WinGetMsg() or WinPeekMsg() is called, the messages are coalesced into a single message.

The value of the QMSG hwnd, pt, and time fields have the values that correspond to the most recent posting. This means that the message is always directed at the last window it was posted to. WinPostQueueMsg() and WinBroadcastMsg() may also be used to post semaphore messages.

The values of all of the lParam1 fields of all semaphore messages posted since the last call to WinGetMsg() or WinPeekMsg() are coalesced by ORing them together. Thus there are 32 bits per semaphore message that may be set individually by different postings, and are cleared only by a call to WinGetMsg() or WinPeekMsg().

The semaphore messages are prioritized in relation to other types of messages as follows:

```
Highest:      WM_SEM1
              Any message in the queue not listed here
             .WM_SEM2
              WM_TIMER
              WM_SEM3
              WM_PAINT
Lowest:       WM_SEM4
```

The value of lParam2 is always 0L.

Semaphore message that are sent via WinSendMsg() are sent exactly like any other message.

The difference between the WM_SEM messages and user registered messages are the following:

1. WM_SEMn messages OR together bits, the others accumulate messages in the queue

2. WM_SEMn messages have more control over priority

3. WM_SEMn messages will not overflow the queue

5.1.2.10 Low level input functions

The following functions aren't typically used by applications. They are typically used by computer-based-training programs, journalling programs, and other system-level applications.

WinGetPhysKeyState

> Format
>
> > ```
> > INT WinGetPhysKeyState(hab, vk)
> > HAB hab;
> > INT vk;
> > ```
>
> Description
> > WinGetPhysKeyState() returns information about the asynchronous (interrupt level) state of the virtual key indicated by vk.
> >
> > The 0x8000 bit is set (less than 0) if the key is down, clear if up. The 0x0001 bit is set if the key has gone down since the last time WinGetPhysKeyState() was called, clear if not. This bit is cleared by a call to WinGetPhysKeyState().
>
> Notes
> > This function returns the physical state of the key; it is not synchronized to the processing of input. See WinGetKeyState().

WinEnablePhysInput

> Format
>
> > ```
> > BOOL WinEnablePhysInput(hab, fEnableInput)
> > HAB hab;
> > BOOL fEnableInput;
> > ```
>
> Description
> > Used to disable queueing of hardware mouse and keyboard events. If fEnableInput is TRUE, mouse and keyboard input are queued as usual. If fEnableInput is FALSE, mouse and keyboard input are disabled. Returns TRUE if input was previously enabled, FALSE otherwise.

5.1.2.10.1 Keyboard State Table

The keyboard state table is a 256 byte table which defines the state of each key the last time a key message a message obtained from the queue was posted. It is indexed by virtual key value. For any virtual key, the 0x80 bit in the corresponding table entry is set if the key is down, 0 if the key is up. The 0x01 bit is set if the key is toggled (pressed an odd number of times), 0 otherwise.

```
typedef UCHAR    KeyStateTable{256};
```

WinSetKeyboardStateTable
_____

Format

```
void WinSetKeyboardStateTable(hab, lpKeyStateTable,
        fSet)
HAB        hab;
UCHAR far *lpKeyStateTable;
BOOL        fSet;
```

Description

Used to get or set the keyboard state table. If fSet is TRUE, sets the keyboard state to *lpbStateTable; if fSet is FALSE, copies the system keyboard state to *lpbStateTable. This function does not change the physical state the keyboard; it changes the state returned by WinGetKeyState(), not WinGetPhysKeyState().

Notes

To set the state of a single key, first get the keyboard state, modify the returned state table, then set the state using the modified tabel.

5.1.3 Window Timers

5.1.3.1 Window Timer Architecture

The window timer functions allow you to cause a message to be posted automatically after a certain amount of time has elapsed.

A timer is identified by a window handle and an ID value. The ID value is a word value specified by the programmer, unless the window handle was specified as NULL. In this case, the ID value is a unique value calculated automatically when the timer is created. Additionally, the timers created by the caret routines and scroll bar routines are given the special ID values of IDCARETTIMER and IDSCROLLBARTIMER respectively. Any windows that may have carets or scroll bars in it must pass any WM_TIMER messages with these IDs to WinDefWindowProc().

A timer can be set to time out in anywhere from 1 to 65536 milliseconds. A timeout value of 0 will cause timers to time out as fast as possible; generally, this is about 1/18th of a second.

When timers time out, a WM_ TIMER message is posted to indicate that the timer has gone off. A timer repeatedly posts WM_ TIMER messages until it is stopped.

WM_ TIMER messages are not actually placed in the message queue. Instead, when WinGetMsg() or WinPeekMsg() is called and there are no other messages in the queue, all started timers are examined to determine if any have timed out. If so, a WM_ TIMER message is returned. WinGetMsg() and WinPeekMsg() check for available timer message before checking for possible WM_ PAINT messages.

1. Timer messages can never fill up a message queue.

2. Timer messages are produced only as often as WinGetMsg() or WinPeekMsg() is called.

3. Timer messages have a lower priority than other queued messages.

4. Timer messages have a higher priority than WM_ PAINT messages.

If more than one timer times out since the last time WinGetMsg() or Win-PeekMsg() was called, the order that the timer messages are received is indeterminate.

If the timer is not associated with a particular window (hwnd == NULL), then WinGetMsg() and WinPeekMsg() will return a WM_ TIMER queue message with hwnd == NULL.

There is a maximum number of timers that can be started in the system. To determine the remaining number of timers that may be started, use the WinGetSysValue() function.

5.1.3.2 Timer Routines

WinStartTimer

Format

```
UINT WinStartTimer(hab, hwnd, idTimer, dtTimeout)
HWND hwnd;
UINT idTimer;
UINT dtTimeout;
HAB hab;
```

Description
This function creates a timer identified by hwnd and idTimer, set to go off every dtTimeout

milliseconds.

A dtTimeout value of 0 will cause the timer to time out as fast as possible; generally, this is about 1/18th of a second.

When a timer times out, a WM_ TIMER message is posted.

If hwnd is not NULL, then WinStartTimer returns 1 (TRUE) if successful, 0 (FALSE) otherwise.

If hwnd is NULL, then the idTimer parameter is ignored, and WinStartTimer returns a unique non-zero ID value that identifies the timer. The timer message is posted in the queue associated with the current thread, with the hwnd field == NULL. Returns 0 if unsuccessful.

A second call to WinStartTimer() for a timer that already exists will reset the existing timer.

WinStopTimer

Format

```
BOOL WinStopTimer(hwnd, idTimer)
HWND hwnd;
UINT idTimer;
HAB hab;
```

Description

This function stops the timer identified by hwnd and idTimer. Returns TRUE if successful, FALSE if the timer didn't exist.

After WinStopTimer() is called, no further messages are received from the stopped timer, even if it has gone off since the last call to WinGetMsg().

5.1.3.3   Timer Messages

WM_ TIMER

Format

```
WM_TIMER
(LOUINT)lParam1: UINT idTimer
lParam2: OL
Returns: OL
```

Description

This message is posted when the timer indicated by idTimer times out. lParam1 contains the ID of the timer that timed out.

This message is always queued. WM_ TIMER messages are treated specially by WinGetMsg() and WinPeekMsg() in a number of ways:

- Timers are processed only by calling WinGetMsg() or WinPeekMsg().

- A timer posts only one WM_ TIMER at a time.

- WM_ TIMER messages have a lower priority than other queued messages.

- WM_ TIMER messages have a higher priority than WM_ PAINT messages.

Chapter 6
Device Contexts

6.1 Device Contexts

A Device Context is the means of writing data to an output device. It is both the device driver, and the physical device (if any) itself.

There are four types of output device, as follows:

- Screen Device Context, used to write to a window on the screen.
- Memory Device Context, into which a bitmap may be selected to be drawn into.
- Metafile Device Context, used to generated a metafile.
- Other device Device Context, used to communicate with a printer, plotter, etc.

A variant on the last of the above types is an 'information' Device Context, used only for querying. A common use for this type of Device Context is for querying information such as font metrics for a particular printer, in order to mimic on the screen the spacing etc of the output data, as it would appear on that printer.

Having created a Device Context, the application can associate it with a presentation space. Drawing into this presentation space then causes output to the associated Device Context.

In some cases, direct output to the Device Context is required, and the DevEscape function is provided for this purpose.

6.1.1 Device Context Functions

DevOpenDC

```
HDC DevOpenDC (hab, type, token, length, data)
HAB hab;
LONG type;
LPSZ token;
LONG length;
LPBUF data;
```

Creates an output Device Context of a specified type.

The data passed depends upon the type of Device Context being created. It provides information such as the driver name, and may also provide data with which the Device Context is to be initialised.

Parameters:

hab   The anchor block handle

type   The type of Device Context to be created, as follows:

1 - OD_ DISPLAY
> The screen device.

2 - OD_ QUEUED
> A device such as a printer or plotter, for which output is to be queued by the spooler.

5 - OD_ DIRECT
> A device such as a printer or plotter. Output is not queued by the spooler.

6 - OD_ INFO
> As OD_ DIRECT, but will only be used to retrieve information (for example, font metrics). Drawing can be performed to a presentation space associated with such a Device Context, but no output medium will be updated.

7 - OD_ METAFILE
> The Device Context will be used to write a metafile. The graphics field defines the area of interest within the picture in the metafile (see the section, "Transform Functions" in the chapter, "Graphics Programming Interface").

8 - OD_ MEMORY
> A Device Context which will be used to contain a bitmap.

token   A string which identifies device information, held in the *PRESSERV.INI* file. This information is the same as that which may be pointed to by *data*; any that is obtained from *data* overrides the information obtained by using *token*.

If *token* is specified as *"" then no device information is taken from *PRESSERV.INI*. Presentation Manager Release 1 will require *"" to be specified.

length The length of *data* supplied. This may be shorter than the full list if omitted items are irrelevant or supplied from *token* or elsewhere.

data   A long pointer to a parameter block containing:-

```
struct DOPDATA
      LPSZ driver_name;
```

```
    LPBUF driver_data;
    LPSZ log_addr;
    LPSZ data_type;
    LPSZ comment;
    LPSZ proc_name;
    LPSZ proc_params;
    LPSZ spl_params;
    LPSZ network_params;
  ;
```

---

driver_ name

> A string containing the name of the device driver (eg "EPSON"). This information must always be supplied if it is not available from *token*.

driver_ data

> Data which is to be passed directly to the device driver. Whether or not any of this is required depends upon the device driver, though the information may alternatively have been specified via DevSetEnvironment.

> The data consists of the following:-

```
struct DRIVDATA
    LONG length;
    LONG version;
    SZ device_name;
    ULONG general_data;
  ;
```

---

length   The length of the whole *driver_ data* structure.

version   The version number of the data. Version numbers are defined by particular device drivers.

device_ name

> A string in a 32-byte field, identifying the particular device (model number etc). Again, valid values are defined by device drivers.

general_ data

> Data as defined by the device driver.

> If the device *type* is OD_ MEMORY, this is a

handle to a Device Context (type HDC), which is compatible with bitmaps which are to be used with this Device Context. If this is not supplied or is null, compatibility with the screen is assumed.

For other device types, the type of *general data* will be defined by the device driver.

log_addr

The logical address of the output device (eg "LPT1").

- For a OD_DIRECT device, this is required if it is not available from *token.*

- For a OD_QUEUED device, this is optional, since the spooler will provide a default if necessary.

data_type

- For a OD_QUEUED device, *data type* defines the type of data which is to be queued, as follows:

  - Q_STD - standard format

  - Q_ESC - escape format

  - Q_RAW - raw format

  Note that a device driver may define other data types. For a full description see the chapter, "The Spooler". If *data type* is not specified for a OD_QUEUED device, the default is supplied by the device driver.

In the above case, *data type* information is defaulted if not specified.

For any other device *type, data type* is ignored.

comment

A natural language description of the file. This may, for example, be displayed for a OD_QUEUED device by the spooler to the end user. For a OD_METAFILE, it is a descriptive record of up to 253 bytes which is

returned on GpiPlayMetaFile. It is optional for any device.

proc_ name
> The name of the queue processor. This is only relevant for a OD_ QUEUED device, and will normally be defaulted.

proc_ params
> A parameter string for the queue processor. This is only relevant for a OD_ QUEUED device, and is optional.

spl_ params
> A parameter string for the Spooler, which is optional. This has the following options, which must be separated by one or more blanks:

> - FORM=f

>   Specifies a forms code 'f'. This must be a valid forms code for the printer (see the section, "Printers" in the chapter, "The Spooler").

>   If not specified, then the data is printed on the forms in use when this print job is ready to be printed.

> - PRTY=n

>   Specifies a priority in the range 0-99, with 99 being the highest. If not specified, then a priority of 50 is used.

network_ params
> network parameters. This is only relevant for a OD_ QUEUED device, and will be defaulted by the spooler if not supplied.

Returns:

```
 0 Error
!=0 Device context handle
```

Principal errors:

```
GPIERR_INVALID_LENGTH
GPIERR_TOKEN_NOT_ASTERISK
Others TBD
```

DevCloseDC

```
HMF DevCloseDC (hdc)
```

```
HDC hdc;
```

This closes the specified Device Context.

It is an error if the Device Context is currently associated with a presentation space. It is also an error if the Device Context was created with WinCreateWindowDC (ie this is a screen Device Context or a micro-PS).

Parameters:

hdc   Specifies the handle for the Device Context

Returns:

```
0 Error
1 OK (not metafile device context)
!=0 Metafile handle (metafile device context)
```

Principal errors:

-

DevPostDeviceModes

```
SHORT DevPostDeviceModes (hab,driver_data,driver_name,
        device_name,log_addr)
HAB hab;
LPBUF driver_data;
LPSZ driver_name;
SZ device_name;
LPSZ log_addr;
```

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 application can call the function 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 DevOpenDC as *driver_ data*.

Parameters:

hab   The anchor block handle

driver_ data

> 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 the same as that defined

for *driver_ data* for DevOpenDC.

driver_ name
> A string containing the name of the device driver

device_ name
> A string in a 32-byte field, identifying the particular device (model number etc). Valid names are defined by device drivers.

log_ addr
> The logical address of the output device (eg "LPT1").

Returns:

```
data pointer was NULL:-
    -1 Error
    0 No settable options
    >0 Size in bytes required for data area
data pointer was not NULL:-
    -1 Error
    0 No device modes
    1 OK
```

Principal errors:

(TBD)

DevEscape

```
LONG DevEscape (hdc, code, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG code;
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function allows applications to access facilities of a device which are not otherwise available through the API. Escape calls are in general sent to the device driver and must be understood by it.

The effects of Escape will be metafiled.

Parameters:

hdc   The handle of the Device Context

code   Specifies the escape function to be performed. The following are currently defied:-

- 1 - QueryEscSupport

- 2 - StartDoc
- 3 - EndDoc
- 4 - NewFrame
- 5 - NextBand
- 6 - AbortDoc
- 7 - DraftMode
- 8 - GetScalingFactor
- 9 - FlushOutput
- 10 - RawData

Devices can define additional escape functions, using *code* values > 32767.

in_count
> The number of bytes of data pointd to by *in_data*.

in_data
> The input data structure required for this escape.

*out_count
> The number of bytes of data pointed to by *out_data*.

out_data
> A buffer which will receive the output from this escape. If *out_data* is null, no data is returned.

Returns:

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

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_LENGTH
```

The following descriptions give the specific syntax and meaning of each *DevEscape* call.

QueryEscapeSupport

```
LONG DevEscape (hdc, QueryEscapeSupport, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function finds out whether a particular escape function is implemented by the device driver. The return value gives the result.

Parameters:

hdc The handle of the Device Context

FlushOutput
  Specifies the escape function to be performed.

in_count
  The number of bytes pointed to by *in_data.*

in_data
  Pointer to an escape code value specifying the escape function to be checked.

*out_count
  Not used, and can be set to zero.

out_data
  Not used, and can be set to null.

Returns:

-1 Error
 0 Escape not implemented for this code
 1 OK

Principal errors:

GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_LENGTH

StartDoc

```
LONG DevEscape (hdc, StartDoc, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function allows an application to specify that a new print job is starting and that all subsequent *NewFrame* calls should be spooled under the same job, until an *EndDoc* call occurs.

This ensures that documents longer than one page are not interspersed with other jobs.

Parameters:

hdc The handle of the Device Context

StartDoc
> Specifies the escape function to be performed.

in_ count
> Specifies the number of characters in the string pointed to by *in_ data*.

in_ data
> Pointer to an ASCIIZ string, specifying the name of the document.

*out_ count
> Not used, and can be set to zero.

out_ data
> Not used, and can be set to null.

Returns:

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

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_LENGTH
```

EndDoc

```
LONG DevEscape (hdc, EndDoc, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function ends a print job started by *StartDoc*.

Parameters:

hdc   The handle of the Device Context

EndDoc   Specifies the escape function to be performed.

in_ count
> Not used, and can be set to zero.

in_ data
> Not used, and can be set to null.

*out_ count
> Not used, and can be set to zero.

out_ data
> Not used, and can be set to null.

Returns:

-1 *Error*
0 *Escape not implemented for this code*
1 *OK*

Principal errors:

GPIERR_INVALID_DC_HANDLE

NewFrame

```
LONG DevEscape (hdc, NewFrame, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function allows an application to specify that it has finished writing to a page. It is similar to *GpiErase* processing for a Screen DC, and causes a reset of the attributes (eg color, mix). This escape is usually used with a printer device to advance to a new page.

Parameters:

hdc   The handle of the Device Context

NewFrame
           Specifies the escape function to be performed.

in_count
           Not used, and can be set to zero.

in_data
           Not used, and can be set to null.

*out_count
           Not used, and can be set to zero.

out_data
           Not used, and can be set to null.

Returns:

-1 *Error*
0 *Escape not implemented for this code*
1 *OK*

Principal errors:

GPIERR_INVALID_DC_HANDLE

NextBand

```
LONG DevEscape (hdc, NextBand, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
```

```
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function allows an application to specify that it has finished writing to a band. The coordinates of the next band are returned. This is used by applications that handle banding themselves (see the section, "Printing Using Banding" in the chapter, "The Spooler").

Parameters:

hdc   The handle of the Device Context

NextBand
> Specifies the escape function to be performed.

in_count
> Not used, and can be set to zero.

in_data
> Not used, and can be set to null.

*out_count
> Specifies the number of bytes of data pointed to by *out_data*. On return, this is updated to the number of bytes actually returned.

out_data
> The address of a buffer which will receive the output from this escape. A structure is returned, containing the device coordinates of the next band, which is a rectangle. The format of the structure is:

```
struct BANDRECT
     LONG xleft;
     LONG ytop;
     LONG xright;
     LONG ybottom;
     ;
```

xleft   The x coordinate of the upper left corner of the rectangular band.

ytop   The y coordinate of the upper left corner of the rectangular band.

xright   The x coordinate of the lower right corner of the rectangular band.

ybottom
> The y coordinate of the lower right corner of the rectangular band.

An empty rectangle (ie xleft $>$ xright, ytop $<$

ybottom) marks the end of the banding operation.

Returns:

-1 Error
0 Escape not implemented for this code
1 OK

Principal errors:

GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_LENGTH

AbortDoc

```
LONG DevEscape (hdc, AbortDoc, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function aborts the current job, erasing everything the application has written to the device since the last *EndDoc*.

Parameters:

hdc   The handle of the Device Context

AbortDoc
            Specifies the escape function to be performed.

in_count
            Not used, and can be set to zero.

in_data
            Not used, and can be set to null.

*out_count
            Not used, and can be set to zero.

out_data
            Not used, and can be set to null.

Returns:

-1 Error
0 Escape not implemented for this code
1 OK

Principal errors:

GPIERR_INVALID_LENGTH

DraftMode

```
LONG DevEscape (hdc, DraftMode, in_count, in_data,
        out_count, out_data)
HDC hdc;
```

61

```
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function turns draft mode on or off. Turning it on instructs the device driver to print faster and with lower quality, if necessary. The draft mode can only be changed at page boundaries (eg after a *NewFrame*).

Parameters:

hdc   The handle of the Device Context

DraftMode
            Specifies the escape function to be performed.

in_count
            Specifies the number of bytes pointed to by *in_data*.

in_data
            A long pointer to a SHORT integer value specifying the mode: 1 for draft mode on, 0 for off.

*out_count
            Not used, and can be set to zero.

out_data
            Not used, and can be set to null.

Returns:

-1 Error
0 Escape not implemented for this code
1 OK

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_LENGTH
```

Note: The default is draft mode off.

GetScalingFactor

```
LONG DevEscape (hdc, GetScalingFactor, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function retrieves the scaling factors for the x and y axes of a printing device. For each scaling factor, an exponent of two is put in *out_data*. Thus, the value 3 is used if the scaling factor is 8.

Scaling factors are used by devices that cannot support graphics at the same resolution as the device resolution.

Parameters:

hdc   The handle of the Device Context

GetScalingFactor
    Specifies the escape function to be performed.

in_count
    Not used, and can be set to zero.

in_data
    Not used, and can be set to null.

*out_count
    Specifies the number of bytes of data pointed to by *out_data*. On return, this is updated to the number of bytes actually returned.

out_data
    The address of a buffer which will receive the output from this escape. A structure is returned, containing the scaling factors for the x and y axes. The format of the structure is:

```
struct SFACTORS
    LONG x;
    LONG y;
    ;
```

x   The x scaling factor, as an exponent of two.

y   The y scaling factor, as an exponent of two.

Returns:

-1 Error
0 Escape not implemented for this code
1 OK

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_LENGTH
```

DevFlushOutput

```
LONG DevEscape (hdc, FlushOutput, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
LPBUF in_data;
LONG *out_count;
```

```
LPBUF out_data;
```

This function flushes any output in the device's buffer.

Parameters:

hdc   The handle of the Device Context

FlushOutput
        Specifies the escape function to be performed.

in_count
        Not used, and can be set to zero.

in_data
        Not used, and can be set to null.

*out_count
        Not used, and can be set to zero.

out_data
        Not used, and can be set to null.

Returns:

-1 Error
 0 Escape not implemented for this code
 1 OK

Principal errors:

```
GPIERR_INVALID_LENGTH
```

DevRawData

```
LONG DevEscape (hdc, RawData, in_count, in_data,
        out_count, out_data)
HDC hdc;
LONG in_count;
LPBUF in_data;
LONG *out_count;
LPBUF out_data;
```

This function allows an application to send data direct to a device driver. For example, in the case of a printer device driver, this could be a printer data stream.

Parameters:

hdc   The handle of the Device Context

RawData
        Specifies the escape function to be performed.

in_count
        The number of bytes pointed to by *in_ data.*

in_ data

>Pointer to the raw data. to be checked.

*out_ count

>Not used, and can be set to zero.

out_ data

>Not used, and can be set to null.

Returns:

-1 Error
0 Escape not implemented for this code
1 OK

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_LENGTH
```

DevQueryHardcopyCaps

```
LONG DevQueryHardcopyCaps (hdc, start, count, info)
HDC hdc;
LONG start;
LONG count;
LPBUF hcinfobuf;
```

This function returns information about the hardcopy capabilities of the device.

hdc Specifies the handle for the DC.

start Specifies which form code number the query is to start from. Used with *count*.

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 *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.

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

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_ARRAY_COUNT
```

DevQueryCaps

```
BOOL DevQueryCaps (hdc, element_no, count, array)
HDC hdc;
LONG element_no;
LONG count;
LONG array[];
```

This function returns information about the capabilities of the device.

Parameters:

hdc   Specifies the handle for the Device Context

element_ no

Gives the index number of the first item of information to be returned in *array*. The first element is number 1.

count   Gives the number of items of information to be returned in *array*

array[count]

An array of *count* elements in which characteristics information is to be returned. The first item returned is set into the first element of the array, the second into the next, and so on.

The following element numbers are defined:-

1   Device family (values as for *type* on DevOpenDC)

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 ∈
         red, green and blue
Bit 1 - 1 if color table with other than 8 b.
         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	Graphics drawing subset supported
30	Graphics architecture version number supported
31	Graphics vector drawing subset supported
32	Device windowing support

Bit 0 - 1 if Device supports windowing

Other bits are reserved zero.

33	Additional graphics support

Bit 0 - 1 if Device supports geometric line t

Other bits are reserved zero.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_ARRAY_COUNT
GPIERR_INVALID_ELEMENT_NUMBER

Chapter 7
Graphics Programming Interface

7.1 Graphics Programming Interface (GPI)

The GPI provides the function for drawing graphics elements on output
devices, including displays, printers, plotters, etc.

7.1.1 GPI Invocation Mechanism

The method of invoking the GPI functions from DOS protected mode
applications is via a set of FAR function calls, linked to the calling code by
means of the DOS dynamic link mechanism. The form of the invocations
and the parameters passed follow the DOS conventions. Function calls are
made directly from the application program to the system functions via
the dynamic link mechanism. The invocation mechanism and the handling
of return codes is similar to that for other DOS function calls, providing a
language-independent interface.

Parameters passed across the API are in Intel format (except where other-
wise stated in the API description). Integer values are passed as 32-bit
integers in Intel format (again except where otherwise indicated). This
includes all co-ordinate values, and transform elements, which are all
currently in fixed format only. (In the Presentation Manager, all integer
values passed across the API must be within the range -32768 to +32767
(signed) or 0 to 65535 (unsigned), except where the description states that
the value is treated as 2 bytes integer + 2 bytes fractional, ie 65536
represents 1.0.)

7.1.2 GPI Presentation Spaces

7.1.2.1 Relationship to Device Contexts

GPI functions operate on GPI presentation spaces. An application may
have multiple GPI presentation spaces, each associated with a different
Device Context (see the chapter "Device Contexts").

GpiCreatePS will create a GPI presentation space and return a GPI handle
which is then used to identify that particular GPI presentation space in
any subsequent GPI function call. Similarly, each Device Context has a
handle which identifies it uniquely. A GPI presentation space is associated
with a particular Device Context using GpiAssociate, which requires both
the GPI presentation space handle and the Device Context handle to be
passed as parameters.

A GPI presentation space consists of the following (where they have been
defined):-

- Segment store

- Definition of symbol sets and fonts

- Definition of line-type sets

- Various controls, eg draw controls

- Logical color table

- Viewing pipeline, down to and including the page and page window

These objects will be retained by the presentation space if it is re-associated with a new Device Context. Thus it is possible to generate and display a picture while the presentation space is associated with a screen Device Context, and then to reassociate the presentation space with a Device Context for a printer or metafile, and redraw it.

In many cases this will produce a good copy of the picture on the new device. There are, however, some potential problem areas:-

- If the devices have markedly different resolution, and 'raster' type operations have been used, for example:-

  - BitBlt, flood fill, set pel

  - Image

  - Image symbol sets or fonts

  - Region drawing or clipping

- If more colors have been used than are available on the new device

- If a loadable color table has been used, and the new device has no facility for loading color tables

- If the drawing area for the new device is less, and the picture (page) was laid out in metric units

- If the aspect ratios (y/x pel spacing) for the two devices are different, and the picture (page) was laid out in pels

- If fonts or symbol sets have been used which are unavailable on the new device

7.1.2.2 Normal PS and Micro-PS

A Gpi presentation space can be defined (at creation) to be one of two types:-

- Normal PS
- Micro-PS

A normal PS is one for which the full range of Gpi functions, as defined in the following sections, is available. A micro-PS supports only a subset of these functions, but in cases where the subset is adequate, provides reduced storage overhead and enhanced performance (the latter especially at association time).

7.1.2.2.1 Normal PS

A normal PS will be used where frequent re-association is not required, or where there is a need for one or more of the functions disallowed in a micro-PS. It is recommended that a normal PS be chosen unless there is a specific reason for requiring a micro-PS.

7.1.2.2.2 Micro-PS

Micro-PS's are suitable for cases where relatively simple drawing is needed to a large number of windows (for example, to implement controls such as button boxes), and where it is undesirable for storage reasons for each window to have a permanently-associated presentation space, yet the execution overhead of frequent associations is also unacceptable.

A micro-PS may not be re-associated with a new Device Context. When it is created, the Device Context with which it is to be associated is specified, and this may not be changed. If, therefore, a picture is to be copied to a metafile, although a micro-PS may be used for this purpose, it will not be the same one as that which was used to draw the picture on the screen.

A micro-PS may be associated with any kind of Device Context. For screen devices, a micro-PS will usually be used in conjunction with a cached DC.

The following Gpi functions are invalid to a micro-PS:-

- GpiAssociate (no *re-* association may be performed)
- Storing of segments (see GpiSetDrawingMode)
- Segment manipulation functions (see the section "Segment Manipulation Functions")
- Passing a buffer of function orders (GpiPutData)
- Segment drawing (GpiDrawChain etc)
- Setting 'push' attribute mode (see GpiSetAttrMode)
- GpiCallSegment, GpiPop
- GpiSetTag, GpiQueryTag

- Structured correlation (eg GpiCorrelateChain) (correlation on individual primitives is allowed by setting the correlate flag in GpiSetDrawControl)

- GpiSetSegmentOrigin, GpiQuerySegmentOrigin

- GpiSetSegmentTransform

Also, less error checking is performed for a micro-PS.

Note that a micro-PS does support loading of symbol sets, logical font definitions, logical color tables, and line type definitions.

7.1.3 Stored and Non-Stored Graphics Output

In the Presentation Manager GPI, graphic primitives and attributes may be

- stored in a segment ('store' mode)

- drawn immediately ('draw' mode)

- both ('draw-and-store' mode)

depending upon the current Drawing Mode (and also on other factors; see GpiSetDrawingMode).

In each case, graphics primitives may be passed across the API either as individual functions such as GpiPolyLine, or as a buffer of orders using GpiPutData.

With immediate drawing, the graphic primitive(s) are drawn on the display surface immediately and the system 'forgets' about the primitive(s) once they are drawn.

In stored mode, the drawing primitives are stored in one or more segments, but not drawn until later (segment drawing requests can be initiated with GpiDrawChain, GpiDrawSegment or GpiDrawFrom).

The composite mode of draw-and-store is provided where the picture primitives are to be drawn as they are passed across the API, but where segments are also to be built for later drawing.

Stored Graphics is good for building complex pictures, for handling graphics databases and for drawing pictures that are drawn many times with only few modifications. It is also useful in relieving the application of the burden of redrawing windows itself if a windowing operation occurs, since the GPI system can handle this.

Non-stored Graphics is good for fast drawing of relatively simple pictures, or where the application wishes to maintain its own graphics database.

In the Functional Descriptions which follow, each function which will cause a drawing order to be constructed and placed in the current segment includes a statement to that effect. Typically these include primitive functions, and attribute functions which change the value of a modal attribute within the picture.

In SAA there will be 2 storage modes, store and non-store. All primitives will be required to be in segments, although unnamed segments (ie ones with an identifier of zero) will be allowed.

In store mode primitives will be placed in segment store and retained, while in non-store mode they will only be kept until they are drawn. Once a non-store segment has been drawn its contents may be deleted, but the current attributes will be retained, as will the fact that a segment is open, so it will not be necessary for the application to re-open a segment or reset the attributes.

The distinction between the two modes will only affect chained segments, unchained segments will be retained but not drawn (until they are called) regardless of the mode selected.

Draw-and-store mode (see GpiSetDrawingMode) is not part of SAA.

7.1.3.1 Stored Graphics Output

Stored Graphics Output functions enable primitives to be stored in Graphics Segments until such time as the segment is destroyed or its contents are overwritten with new data. Each stored segment has a unique name and a set of properties (visible, pickable, dynamic etc.) in addition to its graphics primitives. Segments may be chained together in a required order and can be called from other segments (and from non-stored primitives). The GPI enables drawing, correlation and boundary computation to be performed on an individual segment, part of the segment chain or the whole chain. In addition, operations such as pan and zoom can be accomplished by manipulating the GPI transforms. Special GPI functions are provided to assist the application with rapid removal and redrawing, in exclusive-OR mode, of 'dynamic' segments.

An application using stored segments may leave the default window procedure to redraw its window from the segments if a system windowing operation occurs.

7.1.3.2   Non-Stored Graphics Output

Non-stored graphics output functions allow primitives to be drawn without first creating a segment to contain them. These are termed non-stored primitives, in contrast to stored primitives which are held in long-lived segments in the segment store.

Non-stored primitives are drawn immediately and do not occupy storage once drawn, in contrast to stored primitives. An application using non-stored primitives must redraw its window if a system windowing operation occurs (if stored primitives are used, the window contents can be constructed from the segments).

Non-stored primitives are either executed directly from application function calls or from buffers of graphics drawing orders. Non-stored segments allow segment properties to be specified for groups of non-stored primitives and allow construction of sections of picture in advance (they can, for example be written to a metafile - see the chapter "Metafile Support").

A non-stored segment is started by issuing GpiOpenSegment when the current drawing mode is set to Draw, after which individual primitive functions or GpiPutData may be issued repeatedly. One may also be started (with a default name of zero) by issuing individual primitive functions or GpiPutData outside a segment. In either case, the current attributes are set to default values. In the former case, relevant segment attributes may be changed with GpiSetSegmentAttrs; in the latter case they may not be changed from their initial default values (see GpiOpenSegment).

7.1.3.3   Selection of Stored or Non-Stored

The following summarises the ways in which an application may choose to specify stored or non-stored.

```
        Drawing Mode = Store                    Drawing Mode = Draw
     ----------------------------          ------------------------------

GpiOpenSegment                           GpiOpenSegment (NS)
         .                                        .
         .                                        .
      GpiLine (S)                              GpiLine (NS)
      GpiPutData (S)                           GpiPutData (NS)
         .                                        .
         .                                        .
         .                                        .
GpiCloseSegment                          GpiCloseSegment
      .                                        .
GpiLine (NS)                              GpiLine (NS)
GpiPutData (NS)          GpiPutData (NS)
```

In the above, (S) indicates that the primitives and attributes are stored in a segment without being drawn at this time (they can be drawn later using, for example, GpiDraw), and (NS) that they are non-stored (drawn and discarded).

In draw-and-store mode, the following occurs:-

- Within a segment bracket: the primitive is drawn immediately, and stored in the current segment.

- Outside a segment bracket: the primitive is drawn immediately, and discarded.

7.1.4 Segment Attributes

'Attributes' in this document normally refers to primitive attributes, for example what color should lines be drawn in. *Segment* attributes are quite different from these.

The following is a list of segment attributes:-

Detectability
This can be used to determine whether a correlation function can be performed on the primitives within the segment. For correlation on stored segments see GpiCorrelateChain. Correlation on primitives as they are passed across the API is controlled by the correlate flag on draw controls (see GpiSetDrawControls).

Visibility
Controls whether or not the primitives are to be drawn on the output medium.

Chained
Controls whether or not the segment is a root segment to be included in the segment drawing chain. In draw or draw-and-store modes a chained segment will be drawn as it is passed across the API, an unchained segment will not.

Unchained segments can only be used if called from another segment.

Dynamic
Controls whether or the segment is to be dynamic, that is, drawn using exclusive-OR, so that it may readily be erased by redrawing it. For more information, see the section "Dynamic Segments".

Only stored segments can be dynamic.

Fast chaining
> Controls whether or not, for a chained segment, the system can assume that all primitive attributes need not be reset to default values before execution of the segment.

Propagate detectability
> Controls whether or not the value of the detectability attribute for a segment should be propagated (forced) to all segments beneath it in the hierarchy.

Propagate visibility
> Controls whether or not the value of the visibility attribute for a segment should be propagated (forced) to all segments beneath it in the hierarchy.

Each of these attributes has a default value, which may be changed by GpiSetInitialSegmentAttrs. This is the set which a newly opened segment will be given (except that a non-stored segment will never be flagged as 'dynamic'). Subsequently, a stored segment's attributes may be changed by GpiSetSegmentAttrs.

For primitives outside segments, there is a fixed set of attributes which can never be changed.

Both sets of values are given in the following table.

	SEGMENT ATTRIBUTE	DEFAULT INITIAL	OUTSIDE SEGMENTS (UNCHANGEABLE)
SWM	Detectability	Not detectable	Detectable
SWM	Visibility	Visible	Visible
M	Highlighting	Not highlit	Not highlit
SWM	Chained	Chained	Chained
M	Contains prolog	No prolog	No prolog
WM	Dynamic	Not dynamic	Not dynamic
WM	Fast chaining	Fast chaining	Fast chaining
WM	Propagate detectability	Propagate	Propagate
WM	Propagate visibility	Propagate	Propagate

— S - Defined for the SAA portable subset

— W - Defined for general Presentation Manager applications

— M - Defined for a 'compatible' PS for the GCP migration bindings' use

7.1.5   Primitive Attributes

There are five groups of primitives. These are

- Line and arc primitives
- Character primitives
- Marker primitives
- Area primitives
- Image primitives

Each group has a set of current primitive attributes, which control how these primitives are drawn. For example, lines and arcs have attributes which include line color, line width, line style, etc.

Primitive attributes are set on a modal basis. Once set, the value applies until that attribute is set to a new value, or reset to its default value (this is the value which it starts with when, for example, the presentation space is first created).

Attributes are reset to their default values at the start of a new segment, whether stored or non-stored (though see 'fast chaining', in the section "Segment Attributes"), and at certain other times.

Note that the default values of attributes are fixed, and may not be changed by the application.

7.1.6   GpiPutData

The address and length of a buffer of orders are passed as parameters. The orders are stored and/or drawn onto the output device and the operation is executed synchronously and may not be paused or stopped.

The current attributes may be updated by orders in the buffer.

So that the application need not parse its buffers in advance, the last order in the buffer may be incomplete. In this case, no drawing process check is raised (as would be the case with an incomplete order in a stored segment). The return code indicates that an incomplete order has been found and a returned parameter contains the offset of the order within the buffer.

The application may then add this partial order to the start of the next buffer before it invokes GpiPutData again.

The result is the same as if the application had parsed the data and had split the data into buffers at order boundaries. However, the application need not understand the format of orders.

This is particularly useful for applications dealing with externally-generated graphics data, such as a host datastream application.

7.1.7 Co-ordinate Spaces

A presentation space typically uses application-convenient co-ordinates. The drawing process must eventually generate device co-ordinates, and it will usually be efficient for it to make the transition from application co-ordinates to device co-ordinates in a single step. Notionally, however, there are additional intermediate co-ordinate spaces. The levels of co-ordinate spaces are as follows:-

1. Application convenient units. For graphics, these are World Co-ordinates, and are the units which are used at the API for primitives such as line, arcs etc. For VIO, application convenient units are character cells.

2. (Graphics only) Model Space, which is arrived at by applying the model transforms to World Co-ordinates. This can be thought of as the space in which the picture is constructed, after applying individual transforms for, say, the four wheels of a car.

3. (Graphics only) Page. This can be thought of as the space in which the complete picture, including any subpicturing, is built up.

4. Device co-ordinates. These are the co-ordinates natural to the device, eg pels on a raster display.

Between each of these levels there is a transform. For graphics, the model transform goes from World Co-ordinates to Model Space, the Window-Viewport transform from Model Space to the Page, and the device transform from the Page to device co-ordinates.

For VIO, there is a single transform from character cells to device co-ordinates.

The application can specify various units for the Page, which cause transforms to be defaulted which will be helpful for some commonly-required cases.

Functions are provided to convert a co-ordinate value between any one space and another.

By default, Gpi spaces are defined so that y increases upwards, and x increases to the right. Transformations may, however, be set by the program to produce other effects. With VIO co-ordinates, the row number increases downwards.

Further details of the graphics model and viewing transforms will be found in the section "Transform Functions".

7.1.8   Fonts and Symbol Sets

Gpi describes the use of symbol sets for three purposes:-

- For drawing character strings
- For drawing markers
- For area shading patterns

Fonts carry much more descriptive information than symbol sets, such as the facename, the font family, the weight, whether it is italic, etc etc, and also several items of dimensional information. This gives the system a much better opportunity to synthesize new fonts, from the definitions at its disposal, according to application requirements.

Fonts and symbol sets may be loaded to a GPI presentation space. Fonts are loaded from files, and symbol sets from application storage. For symbol sets an 8-character name is supplied by the application, and this is held by the system as the equivalent of the font facename.

Although the definitions are loaded to the GPI presentation space, they may be suitable only for certain devices (for example, devices with widely different resolutions will require different definitions for a raster 12-point font). It may therefore be necessary to replace them if the presentation space is associated with a new Device Context.

Both image (raster) and vector formats are supported. Proportional spacing and kerning (the latter for fonts only) are also supported.

In addition to any symbol sets / fonts loaded by the application, the system has others permanently loaded and available for use.

Before invoking any kind of draw operation which will require the use of a symbol set or font, the application must issue a select function, passing a list of the required attributes for the symbol set / font, and the local id (lcid) by which it will refer to it later. At this point the system tries to match the required attributes with the definitions available to it, and either selects for use one of the sets of definitions, or synthesizes a new set based on one of the ones available. Synthesis includes scaling image definitions, converting normal weight to bold, italicising, etc.

In scanning the definitions available, the system assigns weights for each attribute mismatch between the requirements and the available definitions. If the application wishes to ensure that a particular set of definitions is selected, it can match its requirements exactly to the attributes of that set - which can be found by a query.

7.1.9 Color

An application may load a logical color table. This identifies the color indices which the application intends to use, and an RGB representation of the colour it would like for each index. In this case the system will translate the color index, as each primitive is drawn, to the index which will give the closest approximation to the required color on the current device.

Normally the index will be an index into the table. An option is provided, however, to allow an application to use RGB values as the color 'index'.

There is a default color table, which defines the colors required for color indices 0 through 7. This will be used for any index within this range, where no logical color table has been defined (or one has, but the index used is outside the range of the one defined).

The logical color table facility is provided to help applications to achieve the best color results on different devices. The logical color table is retained in the presentation space, so that a new translation will be performed automatically if the presentation space is associated with a Device Context for a different device. It is also transmitted in a metafile.

The function of loading a *physical* color table to the device (if it supports this) is a different operation (see the Escape function), and is one which should not normally be performed by an application to a shared device (eg the screen).

A function is, however, provided for an application to request that the physical color table be updated so as to give the best possible match to its logical color table. Since this might mean that other applications would if visible take on a strange appearance, this function should only be used when an application has been maximised. A corresponding function is provided, which should be issued when the application ceases to be maximised, to cause the default physical palette to be reset.

Note that index translation means that the indices generated by certain mix modes (eg OR) will depend on the translation, so applications should only use such mixes with caution if they depend upon the resulting color to be a specific shade.

See the section "Color and Mix Functions" for more details on color.

7.1.10 Dynamic Segments

If you want to be able to move or change part of the picture very quickly (for example, when dragging part of it with the mouse), then the *dynamic* segment attribute may be useful. Dynamic segments are always drawn in exclusive-OR mode, whatever GpiSetMix functions they contain. This means that, while some visual fidelity may be sacrificed, they can be erased completely from the display simply by redrawing them, providing, of course, that no non-dynamic drawing has taken place in the meantime to the same area of the window.

Having set up a dynamic segment (preferably at the start of the segment chain), it can be drawn by one of two techniques:-

1. By setting the 'draw dynamics' draw control and issuing, for example, a GpiDrawChain, in which case the dynamic segment(s) will be drawn after the non-dynamic segments, or

2. GpiDrawDynamics, which just draws the dynamic segment(s).

To make a change to a dynamic segment,

1. Issue GpiRemoveDynamics, which removes the image of the segment from the display.

2. Change the segment(s), for example with GpiSetSegmentTransform, or by using the editing functions.

3. Issue GpiDrawDynamics to replace the image of the segment on the display.

If there is more than one dynamic segment visible, but not all are to be changed, the *name* range on GpiRemoveDynamics may be used to ensure that only the required one(s) are removed. The subsequent GpiDrawDynamics will automatically only replace the same range.

A GpiDrawDynamics function initiated on another thread can be interrupted by setting the 'stop draw' condition (see GpiSetStopDraw). This can be done if a new mouse position is detected, in order to respond more rapidly to the new position. In this case, GpiRemoveDynamics will know just how much of the dynamic segment(s) need to be 'drawn' in order to erase them.

Dynamic segments may be used even if the rest of the picture is non-stored, providing the application ensures that no non-dynamic drawing occurs over any dynamic segments which are currently visible.

If a presentation space is to be dissociated from a screen Device Context into which dynamic segments have been drawn, the dynamic segments should first be removed. If they are not removed, then after any subsequent re-association, they will no longer be removable by

GpiRemoveDynamics.

7.1.10.1   Errors and Return Codes

A return code is returned for each GPI function.

If this indicates that an error has occurred, then the application may determine the value of the error code by invoking the WinGetLastError function.

The error strategy for the GPI is as follows:-

1. Sufficient validation to avoid a malfunction will always be performed.

2. For environment/objects/resources e.g. SymbolSets, Fonts, Bitmaps, Regions, Segments full error checking (as defined for that function) is performed.

3. For segment drawing, and drawing primitives and primitive attributes in draw mode, error checking is permissive i.e. it is optional whether an invalid value is defaulted or produces the specified error. Essential context checking will, however, be performed.

4. When storing in segment store or metafiling, full checking is performed and all defined errors will be raised.

7.1.10.2   Errors

The following errors are valid on many GPI calls, and are not detailed under individual calls.

GPIERR_ GPI_ BUSY
> All functions with hgpi·as a parameter (except stop draw).

GPIERR_ MATRIX_ OVERFLOW
> All functions that may result in matrix computation.

GPIERR_ INSUFFIENT_ MEMORY
> All functions that result in memory allocation.

GPIERR_ INVALID_ GPI_ HANDLE
> All functions with hgpi as a parameter.

GPIERR_ INVALID_ COORDINATE
> All functions with coordinates as parameters.

GPIERR_ DOS_ ERROR (unexpected DOS error)
> All functions that directly or indirectly issue DOS calls.

Drawing Process Check Errors
> All functions that perform segment drawing/correlation

Metafile recording errors
> All functions that perform metafile recording.

7.1.11   Control Functions

GpiCreatePS

```
HPS GpiCreatePS (hdc, width, height, options)
HANDLE hdc;
LONG width;
LONG height;
ULONG options;
```

Creates a GPI presentation space and returns the GPI handle. An initial association of the new presentation space with a Device Context may also be performed (this is mandatory for a micro-PS).

The GPI handle returned is used on subsequent GPI calls to identify the particular GPI presentation space required.

This call also specifies the size and units of the page in which the picture will be created. See the section "Transform Functions" for more information.

There are two types of Gpi presentation spaces: micro-PS, and normal. Only a restricted subset of functions is allowed to a micro-PS; the storage and execution overheads are, however, reduced. For more details, see the chapter, "Graphics Programming Interface".

A Gpi presentation space may be specified to be in implicit draw mode. In this mode, the drawing mode (see GpiSet-DrawingMode) is controlled automatically by the system, which attempts to keep the device up to date with the contents of the presentation space, without the application having to issue explicit Draw functions. For more details, see the chapter, "Graphics Programming Interface".

Parameters:

hdc   The handle of a Device Context with which the new presentation space is to be associated, if the *associate* flag is set. If this flag is not set, *hdc* must be the anchor block handle; in this case no initial association is performed. For a micro-PS, *associate* must be set, and *hdc* must refer to a Device Context.

width,height
> Give the size of the page

options
> This contains 32 bits (with bit 0 the least significant), in standard Intel format.

> The bits have the following meanings:-

(Bits 0-1) Reserved
> Must be B'00'.

(Bits 2-7) Units
> Indicates the units for the page size. Possible values are

> > PU_ISOTROPIC (B'000001')
> > > Arbitrary units, with the origin at the bottom left.

> > PU_PELS (B'000010')
> > > Pel co-ordinates, with the origin at the bottom left.

> > PU_LOWMETRIC (B'000011')
> > > Units of 0.1 mm, with the origin at the bottom left.

> > PU_HIMETRIC (B'000100')
> > > Units of 0.01 mm, with the origin at the bottom left.

> > PU_LOENGLISH (B'000101')
> > > Units of 0.01 in, with the origin at the bottom left.

> > PU_HIENGLISH (B'000110')
> > > Units of 0.001 in, with the origin at the bottom left.

> > PU_TWIPS (B'000111')
> > > Units of 1/1440 in, with the origin at the bottom left.

> Other values are reserved.

(Bits 8-11) format
> Indicates options to be used when storing co-ordinate values internally in the segment store.

> For most functions, the format is not directly visible to an application. It is, however, visible during editing (eg GpiQueryElement). The format will also have an effect on the amount of storage required for segment store.

format is one of the following:-

GPIF_DEFAULT (B'0000')
>   Default local format (as
>   GPIF_SHORT for Presenta-
>   tion Manager)

GPIF_SHORT (B'0010')
>   2-byte integers

GPIF_LONG (B'0011')
>   4-byte integers

Other values are reserved.

(Bit 12) type
>   Indicates the type of Gpi presentation
>   space required. Note that *associate* must
>   also be set if *type* is set):-

GPIT_NORMAL (B'0')
>   Normal PS

GPIT_MICRO (B'1')
>   Micro-PS

(Bit 13) mode
>   Indicates whether the Gpi presentation
>   space is to operate in implicit draw mode
>   or not, as follows:-

GPIM_NORMAL (B'0')
>   Normal mode

GPIM_IMPLICIT_DRAW (B'1')
>   Implicit draw mode

(Bit 14) associate
>   Indicates whether an implicit association
>   is required between the Gpi presentation
>   space and a specified Device Context:-

GPIA_NOASSOC (B'0')
>   No association is required

GPIA_ASSOC (B'1')
>   Association with *hdc* required

All other bits are reserved and must be B'0'.

Returns:

0 Error
!=0 The new Gpi handle

Principal errors:

```
GPIERR_INVALID_PS_DIMENSION
GPIERR_WIDTH_OR_DEPTH_TOO_BIG
GPIERR_INVALID_OR_INCOMPAT_OPTIONS
```

GpiQueryPS

```
ULONG GpiQueryPS (hgpi, width, height)
HPS hgpi;
LONG width;
LONG height;
```

Returns the page parameters, as specified on GpiCreatePS. See GpiCreatePS for details.

Note: bit 16 (associate) of *options* is reserved on GpiQueryPS, and is not necessarily the same as was specified when the presentation space was created.

Parameters:

hgpi   Specifies the handle of the GPI presentation space.

*width,*height
              Variables in which the width and height of the page are returned.

Returns:

```
0 Error
>0 Options (see GpiCreatePS for details)
```

Principal errors:

-

GpiDestroyPS

```
BOOL GpiDestroyPS (hgpi)
HPS hgpi;
```

Destroys the presentation space for the GPI identified by the specified handle. All resources owned by this presentation space are released, and any subsequent calls to the GPI using this handle will be rejected.

Parameters:

hgpi   Specifies the handle of the GPI presentation space.

Returns:

```
0 Error
1 OK
```

Principal errors:

-

GpiResetPS

```
BOOL GpiResetPS (hgpi, options)
HPS hgpi;
ULONG options;
```

This resets the GPI presentation space.

Three levels of reset are provided. These are, in increasing order of power:-

- The equivalent of a segment boundary,

- As if the presentation space had just been created, but without deleting any resources,

- As if the presentation space had just been created.

More details are provided below.

Note that none of these options causes any drawing or erasure to take place on the device. GpiErase may be used to accomplish this. Nor is any association between the specified presentation space and a Device Context affected.

Parameters:

hgpi   Specifies the handle for the GPI presentation space.

options   Flags which control the extent of the reset, as follows:-

GRES_ ATTRS (bit 0)
Set to B'1' (or forced) will cause the equivalent of a root segment boundary. The following will occur:-

- Current attributes are reset to default values.

- Current model transform is reset to unity.

- Current position is set to (0,0).

- Any open clip, stroke, area, defaults, or element brackets are terminated.

- Any currently open segment is closed.

- The current clip area and viewing limits are reset to no clipping.

GRES_ SEGMENTS (bit 1)
Set to B'1' (or forced) will force GRES_ ATTRS, and also:-

- Any stored segments are deleted.

- Initial segment attributes are reset to their initial values

- Default viewing transform, window, viewport, page window and graphics field are reset to the default values

- Drawing mode, draw controls, edit mode, and attribute mode are reset to default values

- Kerning enablement is reset to default values

- Bounds and correlate data are reset

- The currently selected clip region, if any, is deselected

GRES_ ALL (bit 2)
Set to B'1' will force GRES_ ATTRS and GRES_ SEGMENTS, and also:-

- Delete any logical fonts, symbol sets, lcids for bitmaps, and linetype sets.

- Reset any loaded logical color table to default.

Other flags are reserved. An application wishing to reset to the initial state may protect against any future flags being defined by setting GRES_ ALL.

Returns:

0 Error
1 OK

Principal errors:

`GPIERR_INVALID_RESET_OPTIONS`

GpiSavePS

```
LONG GpiSavePS (hgpi)
HPS hgpi;
```

Saves various features of the presentation space on a LIFO stack for the specified presentation space. This stack is different from that which is used for saving attribute values (see the section, "General Attribute Functions"), and GpiSavePS and GpiRestorePS may be used with a micro-PS as well as a normal PS.

The presentation space itself is unchanged.

The following are saved:-

- Current attributes

- Current transforms and clip window and clip area

- Current position

- Reference to selected clip region

- Any loaded logical color table

- References to any loaded logical fonts

- References to any loaded symbol sets

- References to any loaded line type set

- References to the regions created on the associated Device Context

The following are not saved:-

- Default attributes

- The visible region

Note that the actual resources which are referenced in a saved PS (eg clip region, logical fonts, symbol sets, line type set, references to regions) are not copied by GpiSavePS; only references to them are copied. They should not therefore be changed.

This function is valid in an open segment bracket, and also within an open element bracket. If it occurs within an open area, clip area, or strokes bracket, then the corresponding GpiRestorePS should take place before the bracket is closed.

Parameters:

hgpi   Specifies the handle of the GPI presentation space.

Returns:

```
0 Error
>1 The identifier for the saved presentation space.
   This may be used on a subsequent GpiRestorePS.
```

Principal errors:

-

GpiRestorePS

```
BOOL GpiRestorePS (hgpi, psid)
HPS hgpi;
LONG psid;
```

Restores the state of the presentation space to that which existed at the time the corresponding GpiSavePS was issued.

It is possible to restore to a saved presentation space which

was not the one most recently saved. In this case, any which are skipped over on the stack are discarded.

It is an error to issue this function in an open segment bracket.

This function is valid in an open segment bracket, and also within an open element bracket. If it occurs within an open area, clip area, or strokes bracket, then the corresponding GpiSavePS should have taken place earlier in the same bracket.

Parameters:

hgpi   Specifies the handle of the GPI presentation space.

psid   Identifies which saved presentation space is to be restored:-

psid > 1

psid must be the identifier of a saved presentation space on the stack. It is an error if it does not exist.

psid = 1

All (any) entries on the stack are deleted. The presentation space is unchanged.

psid = 0

Is an error. (This might have resulted from an erroneous GpiSavePS.)

psid < 0

The absolute value of psid indicates how many saved presentation spaces back on the stack is required. Thus -1 means that the most recently saved one is to be restored. It is error if the absolute value is larger than the number of entries on the stack. space is returned. This may be used on a subsequent GpiRestorePS.

If an error is returned, the stack is unchanged, as is the current presentation space.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_ID

GpiAssociate

```
BOOL GpiAssociate (hgpi, hdc)
HPS hgpi;
HANDLE hdc;
```

Associate a GPI presentation space with a Device Context. Any type of Device Context may be used. Subsequent stored or non-stored drawing functions direct output to this Device Context.

If a null handle is supplied for the Device Context, the presentation space is just dissociated from the currently associated Device Context.

If, however, the Device Context handle is not null, then it is an error if either the presentation space is currently associated with another Device Context, or the Device Context is currently associated with another presentation space.

The processing described for GRES_ ATTRS (see GpiResetPS) is performed on the presentation space. In addition, bounds and correlate data are destroyed, and any selected clip region is lost.

Any dynamic segments left drawn on the device will not be subsequently removable by GpiRemoveDynamics.

Parameters:

hgpi   Specifies the handle of the GPI presentation space. This must be a normal PS.

hdc   Specifies the handle of the display context. If null, a dissociation occurs.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_PS_ALREADY_ASSOCIATED
GPIERR_DC_ALREADY_ASSOCIATED
```

GpiErrorSegmentData

```
LONG GpiErrorSegmentData (hgpi, name, context)
HPS hgpi;
LONG *name;
LONG *context;
```

A function that returns information about the last error that occurred during a segment drawing operation.

The information returned is the segment name, the context,

and the byte offset or element number, depending upon the context.

The byte offset is returned for the following contexts:-

- The error occurred within the data of a GpiPutData, or

- The error occurred within the data of a GpiElement.

Otherwise (segment context), the element number is returned.

Parameters:

hgpi Specifies the handle for the GPI presentation space.

*name A variable in which the segment name is to be returned.

*context

A variable in which the context of the error is returned:-

GPIE_ SEGMENT (0)
The error occurred while processing the contents of a stored segment

GPIE_ ELEMENT (1)
The error occurred while processing the contents of a GpiElement

GPIE_ DATA (2)
The error occurred while processing the contents of a GpiPutData

Returns:

-1 Error
>=0 Position. This is either the byte offset or the element
number, depending upon*context* (see above).

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION

7.1.12 Drawing Functions

GpiErase

```
BOOL GpiErase (hgpi)
HPS hgpi;
```

Clears the output display of the Device Context associated with the specified GPI presentation space, to the zero color

index value.

This operation is independent of the settings of the draw controls (see GpiSetDrawControl), and also of any application clipping which may be in force.

If this function is followed by primitives or attributes, without first opening a segment, then the processing will be as described for GpiCloseSegment.

Parameters:

hgpi   Specifies the handle for the GPI presentation space.

Returns:

0 Error
1 OK

Principal errors:

-

GpiSetDrawControl

```
BOOL GpiSetDrawControl (hgpi, control, value)
HPS hgpi;
LONG control;
LONG value;
```

This function sets various options for subsequent GpiDraw... and GpiDrawDynamics drawing operations.

The default values are off for all controls other than *Display*, which is on.

It is an error to issue this function in any of the following cases:-

● Inside an open segment

● Outside an open segment, but inside one of the following:-

    ● Area bracket

    ● Strokes bracket

    ● Element bracket

    ● Clip area bracket

Parameters:

hgpi   Specifies the handle for the GPI presentation space.

control   Specifies which drawing control is to be changed, as follows:-

1 - Erase before draw

> Before GpiDrawChain, GpiDrawFrom, or GpiDrawSegment, perform an implicit GpiErase operation.

2 - Display (*)

> Allow drawing to take place on the output medium.

> If this flag is off, then except for GpiErase, no output operations appear on the output medium. This includes raster operations, drawing primtives, GpiDraw operations, etc.

3 - Accumulate boundary data (*)

> During any output operations except GpiErase, accumulate the bounding rectangle of the drawing. See the section, "Correlation and Boundary Determination Functions".

4 - Draw dynamic segments

> Perform an implicit GpiRemoveDynamics before GpiDrawChain, GpiDrawFrom, or GpiDrawSegment, and an implicit GpiDrawDynamics afterwards.

5 - Correlate (*)

> When GpiPutData, GpiElement, or individual drawing primitives are passed across the API, perform a correlation operation on them, and set a return code if a hit occurs.

Controls identified by (*) above are the only ones relevant to a micro-PS. Any other control settings will be ignored for a micro-PS.

value   Specifies the required value of the drawing control:-

> *0* Off
> *1* On

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_SEG_CONTEXT_ERROR
GPIERR_ELEMENT_CONTEXT_ERROR
GPIERR_AREA_CONTEXT_ERROR
GPIERR_CLIP_AREA_CONTEXT_ERROR
GPIERR_STROKES_CONTEXT_ERROR
GPIERR_INVALID_DRAW_CONTROL
GPIERR_INVALID_DRAW_VALUE
```

GpiQueryDrawControl

```
LONG GpiQueryDrawControl (hgpi, control)
HPS hgpi;
LONG control;
```

This returns a drawing control set by GpiSetDrawControl.

Parameters:

hgpi   Specifies the handle for the GPI presentation
         space.

control identifies the control whose value is to be returned,
         as follows:-

> *1* Erase before draw
> *2* Display
> *3* Accumulate boundary data
> *4* Draw dynamic segments
> *5* Correlate

Returns:

-1 Error
>=0 Value of the control. See GpiSetDrawControl for details.

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_DRAW_CONTROL
```

GpiDrawChain

```
BOOL GpiDrawChain (hgpi)
HPS hgpi;
```

Draws the picture chain.

The drawing operation is controlled by the functions set by
the draw controls (see GpiSetDrawControl), except for the
correlate control.

If there is not a segment open at the time of the draw, then
at the completion of the draw, processing equivalent to that
described for GpiCloseSegment will be performed. If, how-
ever, a segment is already open at the time of the draw, then
GpiCloseSegment processing will not be performed at the

completion of the draw. In this case, if the open segment is the last in the chain (and no dynamic segments had to be drawn), then attributes etc will be in the correct state to continue drawing in any drawing mode.

It is an error to issue this function while any of the following brackets is open:-

- Area bracket

- Clip area bracket

- Strokes bracket

- Element bracket

Any such error will be detected prior to performing any erase required by the setting of the 'erase before draw' draw control (see GpiSetDrawControl).

Parameters:

hgpi   Specifies the handle for the GPI presentation
            space.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_ELEMENT_CONTEXT_ERROR
GPIERR_AREA_CONTEXT_ERROR
GPIERR_CLIP_AREA_CONTEXT_ERROR
GPIERR_STROKES_CONTEXT_ERROR
GPIERR_STOP_DRAW_OCCURRED (warning)
GPIERR_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_STROKES_DEFN_NOT_COMPLETE (at segment end)
```

Note that for a 'compatible' PS, the return value is SHORT, and can be 0 or 1, as above, and also 2 which means 'correlate hit(s)'.

GpiDrawFrom

```
BOOL GpiDrawFrom (hgpi, name1, name2)
HPS hgpi;
LONG name1;
LONG name2;
```

Draws a section of the picture chain.

Drawing starts at the segment identified by *name1* and includes all chained and called segments up to, and including, the segment identified by *name2*.

The drawing operation is controlled by the functions set by
the draw controls (see GpiSetDrawControl), except for the
correlate control.

If there is not a segment open at the time of the draw, then
at the completion of the draw, processing equivalent to that
described for GpiCloseSegment will be performed. If, how-
ever, a segment is already open at the time of the draw, then
GpiCloseSegment processing will not be performed at the
completion of the draw. In this case, if the open segment is
the last one drawn (and no dynamic segments had to be
drawn), then attributes etc will be in the correct state to
continue drawing in any drawing mode.

It is an error to issue this function while any of the following
brackets is open:-

- Area bracket
- Clip area bracket
- Strokes bracket
- Element bracket

If the 'from' segment does not exist, or is not in the segment
chain, an error is raised. If the 'to' segment does not exist, or
is not in the chain, or is chained before the 'from' segment,
no error is raised, and processing continues to the end of the
chain.

Any errors will be detected prior to performing any erase
requireed by the setting of the 'erase before draw' draw con-
trol (see GpiSetDrawControl).

Parameters:

hgpi	Specifies the handle for the GPI presentation space.
name1	Specifies the first segment to be drawn. It must be > 0.
name2	Specifies the last segment to be drawn. It must be > 0.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_ELEMENT_CONTEXT_ERROR
GPIERR_AREA_CONTEXT_ERROR
GPIERR_CLIP_AREA_CONTEXT_ERROR
```

```
GPIERR_STROKES_CONTEXT_ERROR
GPIERR_STOP_DRAW_OCCURRED (warning)
GPIERR_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_STROKES_DEFN_NOT_COMPLETE (at segment end)
GPIERR_NAMED_SEG_DOES_NOT_EXIST (I.E. NAME1)
GPIERR_NAMED_SEG_NOT_CHAINED (I.E. NAME1)
GPIERR_INVALID_SEG_ID (I.E. NAME1 or NAME2)
```

Note that for a 'compatible' PS, the return value is SHORT, and can be 0 or 1, as above, and also 2 which means 'correlate hit(s)'.

GpiDrawSegment

```
BOOL GpiDrawSegment (hgpi, name)
HPS hgpi;
```

Draws the specified segment.

The drawing operation is controlled by the functions set by the draw controls (see GpiSetDrawControl), except for the correlate control.

If there is not a segment open at the time of the draw, then at the completion of the draw, processing equivalent to that described for GpiCloseSegment will be performed. If, however, a segment is already open at the time of the draw, then GpiCloseSegment processing will not be performed at the completion of the draw. In this case, if the open segment is the last one drawn (and no dynamic segments had to be drawn), then attributes etc will be in the correct state to continue drawing in any drawing mode.

It is an error to issue this function while any of the following brackets is open:-

- Area bracket

- Clip area bracket

- Strokes bracket

- Element bracket

If the 'from' segment does not exist, or is not in the segment chain, an error is raised. If the 'to' segment does not exist, or is not in the chain, or is chained before the 'from' segment, no error is raised, and processing continues to the end of the chain.

Any errors will be detected prior to performing any erase requireed by the setting of the 'erase before draw' draw control (see GpiSetDrawControl).

Parameters:

hgpi   Specifies the handle for the GPI presentation space.

name   Specifies the segment that is to be drawn. It must be > 0.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_ELEMENT_CONTEXT_ERROR
GPIERR_AREA_CONTEXT_ERROR
GPIERR_CLIP_AREA_CONTEXT_ERROR
GPIERR_STROKES_CONTEXT_ERROR
GPIERR_STOP_DRAW_OCCURRED (warning)
GPIERR_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_STROKES_DEFN_NOT_COMPLETE (at segment end)
GPIERR_NAMED_SEG_DOES_NOT_EXIST
GPIERR_INVALID_SEG_ID (I.E. NAME)
```

Note that for a 'compatible' PS, the return value is SHORT, and can be 0 or 1, as above, and also 2 which means 'correlate hit(s)'.

GpiSetStopDraw

```
BOOL GpiSetStopDraw (hgpi, value)
HPS hgpi;
```

This either sets or clears the 'stop draw' condition. While this condition exists, if one of the following operations is either started or already in progress (initiated from another thread), to the specified GPI presentation space, then it is terminated.

The operations are:-

— GpiDrawChain

— GpiDrawFrom

— GpiDrawSegment

— GpiDrawDynamics

— GpiPutData

— GpiPlayMetaFile

The stopped operation will terminate with an error return code.

This function allows an application to set up and control an asynchronous thread, on which long drawing operations may be done. At the point at which the controlling thread

105

realises it wishes to stop a draw, it sets the 'stop draw' condition, and clears it after it has received an acknowledgment from the drawing thread.

The 'stop draw' condition has no effect on any other functions.

(Any operation other than GpiSetStopDraw, directed at a presentation space which is currently in use, will give an error return code, except for a presentation space in implicit draw mode.)

Note that if this function is issued when an asynchronous draw to a metafile is taking place, an unusable metafile will result.

Parameters:

hgpi Specifies the handle for the GPI presentation space.

value The required value of the attribute, as follows:-

```
0 Clear the 'stop draw' condition
1 Set the 'stop draw' condition
```

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_VALUE
```

GpiQueryStopDraw

```
LONG GpiQueryStopDraw (hgpi)
HPS hgpi;
```

This returns an indication of whether the 'stop draw' condition currently exists. See GpiSetStopDraw for details.

Parameters:

hgpi Specifies the handle for the GPI presentation space.

Returns:

```
-1 Error
 0 No 'stop draw' condition currently exists.
 1 The 'stop draw' condition does currently exist.
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
```

GpiRemoveDynamics

```
BOOL GpiRemoveDynamics (hgpi, name1, name2)
HPS hgpi;
LONG name1;
LONG name2;
```

Removes those parts of the displayed image that were drawn from the dynamic segments in a section of the picture chain. This includes any parts that were drawn by calls from these dynamic segments.

The section of the picture chain is identified by the name of the first and last segments in the section. If *name1* and *name2* have the same value, GpiRemoveDynamics erases only the parts drawn from the named segment and by calls from that segment.

GpiRemoveDynamics usually indicates (1) that a dynamic segment is about to be updated; and (2) that, having completed the update, GpiDrawDynamics will be called to redraw the dynamic segments.

If a temporary re-association is to be done, GpiRemoveDynamics should be issued to remove the dynamic segments from the display before the first dissociation.

If this function is followed by primitives or attributes, without first opening a segment, then the processing will be as described for GpiCloseSegment.

If the 'from' segment does not exist, or is not in the segment chain, no action is taken. If the 'to' segment does not exist, or is not in the chain, or is chained before the 'from' segment, no error is raised, and processing continues to the end of the chain.

Parameters:

hgpi   The handle of the GPI presentation space.

name1   The name of the first segment in the section. It must be > 0.

name2   The last segment in the section. It must be > 0.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_ELEMENT_CONTEXT_ERROR
GPIERR_AREA_CONTEXT_ERROR
GPIERR_CLIP_AREA_CONTEXT_ERROR
```

```
GPIERR_STROKES_CONTEXT_ERROR
GPIERR_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_STROKES_DEFN_NOT_COMPLETE (at segment end)
GPIERR_INVALID_SEG_ID (I.E. NAME1 or NAME2)
GPIERR_INVALID_METAFILE_FUNCTION (warning)
```

GpiDrawDynamics

```
BOOL GpiDrawDynamics (hgpi)
HPS hgpi;
```

Redraws the dynamic segments in, or called from, the picture chain. If there is no range set by a previous GpiRemoveDynamics all dynamic segments are redrawn. However, if GpiRemoveDynamics specified a range in the picture chain, the redraw is restricted to the dynamic segments that are in, or called from, the selected range. (See GpiRemoveDynamics)

Note that the redraw is controlled by the functions set by previous calls to GpiSetDrawControl.

Note that the 'stop draw' condition can be set (from another thread) while GpiDrawDynamics is in progress. This is useful in responding to a new position by setting this condition, and then clearing it and redrawing at the new position.

If erase was specified in the most recent call to GpiSetDrawControl, the presentation space is erased before the redraw.

If this function is followed by primitives or attributes, without first opening a segment, then the processing will be as described for GpiCloseSegment.

Parameters:

hgpi   Specifies the handle for the GPI presentation space.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_STOP_DRAW_OCCURRED (warning)
GPIERR_ELEMENT_CONTEXT_ERROR
GPIERR_AREA_CONTEXT_ERROR
GPIERR_CLIP_AREA_CONTEXT_ERROR
GPIERR_STROKES_CONTEXT_ERROR
GPIERR_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_STROKES_DEFN_NOT_COMPLETE (at segment end)
```

GPIERR_INVALID_METAFILE_FUNCTION (warning)

GpiSetDrawingMode

```
BOOL GpiSetDrawingMode (hgpi, mode)
HPS hgpi;
LONG mode;
```

This sets the Drawing Mode to control how subsequent individual drawing order and GpiPutData requests are handled. The orders may be drawn immediately in non-stored mode, and/or stored in the current segment.

Note that any drawing orders which occur *outside* a segment (ie outside a GpiOpenSegment - GpiCloseSegment bracket) are treated as non-stored. Conversely, any segments which are not chained are always stored. The following table summarizes the possibilities:-

Drawing Mode	Type of Segment		
	Chained Segment	Unchained Segment	Outside Segment
DrawAndStore	DrawAndStore	Store	Draw
Store	Store	Store	Draw
Draw	Draw	Store	Draw

The actual drawing mode (referred to when describing other Gpi functions) therefore depends upon the mode as set by GpiSetDrawingMode, in conjunction with the type of segment, as in the table.

It is an error to attempt to set the drawing mode within a segment bracket, and also, outside a segment bracket, in a

- Area bracket
- Strokes bracket
- Clip area bracket
- Element bracket

The default Drawing Mode is Draw (non-stored).

This function is invalid for a presentation space operating in implicit draw mode (see GpiCreatePS).

Parameters:

hgpi   Specifies the handle for the GPI presentation space.

mode   Specifies the mode to be used for subsequent drawing functions, as follows:-

> *1* `Draw (non-stored)`
> *2* `Store`
> *3* `Draw-and-store`

Returns:

> *0* `Error`
> *1* `OK`

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_IMPLICIT_DRAW_FUNCTION
GPIERR_SEG_CONTEXT_ERROR
GPIERR_ELEMENT_CONTEXT_ERROR
GPIERR_AREA_CONTEXT_ERROR
GPIERR_CLIP_AREA_CONTEXT_ERROR
GPIERR_STROKES_CONTEXT_ERROR
GPIERR_INVALID_DRAWING_MODE
```

GpiQueryDrawingMode

```
LONG GpiQueryDrawingMode (hgpi)
HPS hgpi;
```

This returns the Drawing Mode.

Parameters:

hgpi   Specifies the handle for the GPI presentation space.

Returns:

> *0* `Error`
> *>0* `Drawing mode. See GpiSetDrawingMode for details.`

Principal errors:

> -

GpiPutData

```
SHORT GpiPutData (hgpi, control, length, data)
HPS hgpi;
LONG control;
LONG *length;
LPBUF data;
```

Passes a buffer of orders which are either to be added to the current segment, and/or drawn without storing them in a segment, depending upon the current drawing mode (see GpiSetDrawingMode), and whether or not the primitives are

within a segment.

If there is an incomplete order at the end of the buffer, then *length* is updated to point to the start of the incomplete order. The application can then concatenate this partial order in front of the next buffer.

This function is valid within an element bracket (see GpiBeginElement). It may contain GpiBeginElement and GpiEndElement orders, so long as these are in the correct sequence with respect to the currently opened segment in segment store.

Note that no co-ordinate conversion is performed by this function. The application must ensure that the co-ordinates within the buffer are in the correct format for the presentation space.

This function is invalid if the editing mode (see GpiSetEditMode) is set to *replace*, and also in *insert* mode if the element pointer (see the section, "Segment Content Manipulation Functions (indirect)") is not pointing to the last element.

For a 'compatible' presentation space, GpiPutData is only supported in *replace* mode.

Parameters:

hgpi The handle for the GPI presentation space.

control Gives the co-ordinate type and format used in the data. This must be

   5 `Self-identifying`

*length A variable which the application sets to the length of order data pointed to by *data*. On return, if an incomplete order occurred, it is updated to the offset of the start of the incomplete order.

   length must not be greater than 63K.

data Specifies the starting address of the order data.

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_LENGTH
GPIERR_DATA_TOO_LONG
GPIERR_INVALID_ORDER
GPIERR_INVALID_EDIT_MODE
```

GPIERR_INVALID_ELEMENT_POINTER

7.1.13 Correlation and Boundary Determination Functions

7.1.13.1 Correlation

A correlation operation is where the application specifies a rectangle, normally a small one centered about the point that the operator was pointing to on the screen, and asks which, if any, primitives cause information to be displayed that rectangle.

The primitives are subject to clipping by GpiSetViewingLimits, and by GpiSetGraphicsField, but not by any lower level clipping arising from device considerations, such as the limits of the screen window.

For stored segments, correlation is performed independently of drawing. Non-stored primitives may optionally be correlated at the same time as drawing (and/or boundary determination).

Only non-dynamic segments, with the *detectable,* attribute, can be correlated upon. This includes both stored and non-stored segments. Primitives outside segments may also be correlated upon.

7.1.13.2 Boundary Determination

Boundary determination is an operation carried out by the system, which tells the application the smallest bounding rectangle of the primitives or segments drawn. This information is useful to an application in deciding which segments need to be redrawn to heal any particular part of the display.

The primitives are not subject to any clipping. The information is returned in Model Space co-ordinates.

Dynamic segments are not included in boundary determination.

7.1.13.3 Functions

A correlation operation may be performed by one of the following methods, depending upon whether the picture is first stored, or not:-

- For an already stored picture:-
  - Issue one of the GpiCorrelate functions

- Inspect the data returned in the parameters

This method of correlation will only correlate on segments with a nonzero identifier (and not called for a segment with a zero identifier), and on primitives for which the current tag is nonzero.

- For a non-stored picture (draw mode), or while creating a picture in draw-and-store mode:-

  - Set the 'correlate' flag (see GpiSetDrawControl)

  - Set the pick aperture (see GpiSetPickAperture)

  - Issue a series of GpiPutData functions or pass individual primitives across the API.

  - Inspect the return code as each GpiPutData or primitive is passed.

This method of correlation is still performed even if the segment id is zero, and/or the primitive tag is currently zero.

A boundary determination operation may be performed as follows:-

- For an already stored picture:-

  - Set the 'accumulate boundary data' flag (see GpiSetDrawControl)

  - Issue one of the GpiDraw functions

  - Inspect the resulting boundary data (see GpiQueryBoundaryData)

- For a non-stored picture (draw mode), or while creating a picture in draw-and-store mode:-

  - Set the 'accumulate boundary data' flag (see GpiSetDrawControl)

  - Set the pick aperture (see GpiSetPickAperture)

  - Issue a series of GpiPutData functions or pass individual primitives across the API.

  - Inspect the resulting boundary data (see GpiQueryBoundaryData)

Note that in the non-stored case (with either GpiPutData or individual primitives), a hit on the perimeter of an area will be returned *before* a hit on the area interior, which occurs on the GpiEndArea function.

For correlation on geometric thick lines, a hit may be recorded on the nominal width of the strokes as they are passed, and then correlation is performed on the whole (set of) strokes at End Strokes time, in a similar manner to areas.

7.1.13.4   Pick Aperture and Tag Functions

GpiSetPickAperture

```
BOOL GpiSetPickAperture (hgpi, options, x, y, w, h)
HPS hgpi;
LONG x;
LONG y;
LONG w;
LONG h;
```

Sets the position and size of the pick aperture, in Model Space, for subsequent non-stored correlation operations. The dimensions of the pick aperture are inclusive.

Parameters:

hgpi   The handle for the GPI presentation space.

options   Specifies how the values *w* and *h* are to be interpreted, as follows:-

> *0 - PICKAP_ DEFAULT* the default,
>   same as PICKAP_SCALED
> *1 - PICKAP_ SCALED* the width and height
>   are set to w and h,
>   respectively, multiplied by their
>   default values.
> *2 - PICKAP_ RECT* the width and height
>   are set to w and h
>   respectively

x,y   The coordinates of the center of the window.

w,h   Depend upon the setting of *options*, as described above.

           In the case of PICKAP_ SCALED (only), the binary point is considered to be between the second and third bytes; thus 65536 represents the value unity.

Returns:

> *0* Error
> *1* OK

Principal errors:

```
GPIERR_CENTRE_OUTSIDE_PAGE (i.e. x, y)
GPIERR_WINDOW_LIMITS_OUTSIDE_PAGE (W, H TOO LARGE)
GPIERR_INVALID_PICK_APERTURE_DIMENSION
```

GpiQueryPickAperture

```
BOOL GpiQueryPickAperture (hgpi, x, y, w, h)
HPS hgpi;
LONG *x;
```

```
LONG *y;
LONG *w;
LONG *h;
```

This returns the position and size of the pick window, in Model Space co-ordinates.

(For a compatible presentation space, they are in GPS co-ordinates.)

Parameters:

hgpi	The handle for the GPI presentation space.
*x,*y	Set to the coordinates of the center of the aperture.
*w,*h	Set to the width and height of the aperture on the x and y axes, respectively.

Returns:

0 Error
1 OK

Principal errors:

-

GpiSetTag

```
BOOL GpiSetTag (hgpi, tag)
HPS hgpi;
LONG tag;
```

Sets the primitive tag to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

This function can be modified to push the old value onto the segment call stack before setting to the new value (see GpiSetAttrMode).

Parameters:

hgpi	The handle for the GPI presentation space.
tag	The new value for the tag.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION

Note: Setting the tag to *0* is a special case. Graphics primitives

cannot be picked if they are assigned a tag of *0*.

GpiQueryTag

```
BOOL GpiQueryTag (hgpi, tag)
HPS hgpi;
LONG *tag;
```

Sets the current or default primitive tag. This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*tag   A variable in which the tag value is returned.

Returns:

 0 Error
 1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
```

7.1.13.5 Correlation Data Functions

GpiCorrelateChain

```
SHORT GpiCorrelateChain (hgpi, ctype, x, y, atype,
        c1, spec_array, c2, depth, seg_array,
        tag_array, num_hits)
HPS hgpi;
LONG ctype;
LONG x;
LONG y;
LONG atype;
LONG c1;
LONG spec_array[];
LONG c2;
LONG depth;
LONG seg_array[];
LONG tag_array[];
LONG num_hits;
```

Performs a correlate operation on the stored segment chain, and returns data for each tagged primitive that intersects the specified aperture. The data returned for each "hit" (or correlation) consists of a set of segment and tag pairs, starting with the correlated one, then the one which called that segment, repeated until the root segment (which was not called by another segment) is reached.

Only primitives with a nonzero tag (see GpiSetTag) in segments

with a nonzero identifier are correlated using this call. Primitives in segments called (to any depth in the hierarchy) from a segment zero are ineligible for correlation.

The depth value specifies the number of sets of segment and tag pairs to be returned for each hit. If the root segment is reached before *depth* values, the remaining values are set to zero. If more than *depth* values are available, only that number are returned.

The draw controls (see GpiSetDrawControl) are ignored by this function.

If this function is followed by primitives or attributes, without first opening a segment, then the processing will be as described for GpiCloseSegment.

Parameters:

hgpi	The handle for the GPI presentation space.
ctype	The type of segments on which correlation is to be performed:-

0 - PICKSEL_ VISIBLE
> Only visible and detectable segments, with nonzero identifiers, are correlated.

1 - PICKSEL_ ALL
> All segments with nonzero identifiers are correlated, regardless of the detectability and visibility attributes of the segments.

x,y	The co-ordinates of the position of the center of the aperture in Model Space
atype	The type of aperture to be used:-

0 - PICKAP_ DEFAULT
> The default; same as 1.

1 - PICKAP_ SCALED
> Scaled pick aperture

> The *spec_ array* parameter must contain a single element, which is a uniform scaling aperture that is applied to the device's default pick aperture. In this case the binary point of the *spec_ array* parameter is considered to be between the second and third bytes; thus 65536 represents the value unity.

2 - PICKAP_ RECT
> Rectangular aperture

The *spec_array* parameter must contain two Model Space values, giving the width and height (respectively) of a rectangular aperture. The center of the rectangle is positioned at the point given by the values in the *x* and *y* parameters.

c1   The number of elements in the *spec_array* parameter

spec_array[c1]
       An array of numbers as defined by the *atype* parameter

c2   The maximum number of hits which can be returned in the *seg_array* and *tag_array* parameters

depth The number of segment and tag pairs to be returned for each hit

seg_array[c2][depth]
       An array of segment identifiers. For each hit, a set of *depth* values are returned.

tag_array[c2][depth]
       An array of primitive tags. For each hit, a set of *depth* values are returned.

*num_hits
       A variable in which the number of hits in the *seg_array* and *tag_array* parameters is returned.

       A 'hit' is an instance of a segment identifier and tag pair for which the primitives lie completely or partially within the specified aperture. Two different primitives in the same segment might have the same tag, and would therefore produce the same hit. This is counted as a single hit; the hit is only recorded once in the *seg_array* and *tag_array* that are returned. The *num_hits* parameter, therefore, returns this distinct number of hits.

       The tables *seg_array* and *tag_array* are set to the hits that are found, up to the maximum defined in the *c2* parameter. Corresponding sets of elements form the 'hit' pairs. The number returned in *num_hits* therefore contains the number of sets of *depth* pairs set if the *c2* parameter is greater than the number of hits detected. The number of elements set in the *seg_array* and *tag_array* parameters is the number returned in *num_hits* multiplied by the *depth*.

       If the same value is returned in the *num_hits* parameter as is specified in the *c2* parameter, there may be yet more hits that cannot be returned in *seg_array* and *tag_array*. If all hits are important, specify arrays that are large enough to contain the maximum number

of hits that are expected.

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_STROKES_DEFN_NOT_COMPLETE (at segment end)
```

GpiCorrelateFrom

```
SHORT GpiCorrelateFrom (hgpi, name1, name2, ctype,
        x, y, atype, c1, spec_array, c2, depth,
        seg_array, tag_array, num_hits)
HPS hgpi;
LONG name1;
LONG name2;
LONG ctype;
LONG x;
LONG y;
LONG atype;
LONG c1;
LONG spec_array[];
LONG c2;
LONG depth;
LONG seg_array[];
LONG tag_array[];
LONG num_hits;
```

Performs a correlate operation on a section of the stored segment chain, starting at the segment identified by *name1*, and including chained and called segments up to, and including, the segment identified by *name2*.

Data is returned for each tagged primitive that intersects the specified aperture. The data returned for each "hit" (or correlation) consists of a set of segment and tag pairs, starting with the correlated one, then the one which called that segment, repeated until the root segment (which was not called by another segment) is reached.

Only primitives with a nonzero tag (see GpiSetTag) in segments with a nonzero identifier are correlated using this call. Primitives in segments called (to any depth in the hierarchy) from a segment zero are ineligible for correlation.

The depth value specifies the number of sets of segment and tag pairs to be returned for each hit. If the root segment is reached before *depth* values, the remaining values are set to zero. If more than *depth* values are available, only that number are returned.

The draw controls (see GpiSetDrawControl) are ignored by this function.

If this function is followed by primitives or attributes, without first opening a segment, then the processing will be as described for GpiCloseSegment.

If the 'from' segment does not exist, or is not in the segment chain, an error is raised. If the 'to' segment does not exist, or is not in the chain, or is chained before the 'from' segment, no error is raised, and processing continues to the end of the chain.

Parameters:

hgpi   The handle for the GPI presentation space.

name1   Specifies the first segment to be correlated. It must be > 0.

name2   Specifies the last segment to be correlated. It must be > 0.

ctype   The type of segments on which correlation is to be performed:-

0 - PICKSEL_ VISIBLE
          Only visible and detectable segments, with nonzero identifiers, are correlated.

1 - PICKSEL_ ALL
          All segments with nonzero identifiers are correlated, regardless of the detectability and visibility attributes of the segments.

x,y   The co-ordinates of the position of the center of the aperture in Model Space

atype   The type of aperture to be used:-

0 - PICKAP_ DEFAULT
          The default; same as 1.

1 - PICKAP_ SCALED
          Scaled pick aperture

          The *spec_ array* parameter must contain a single element, which is a uniform scaling aperture that is applied to the device's default pick aperture. In this case the binary point of the *spec_ array* parameter is considered to be between the second and third bytes; thus 65536 represents the value unity.

2 - PICKAP_RECT
Rectangular aperture

The *spec_ array* parameter must contain two Model Space values, giving the width and height (respectively) of a rectangular aperture. The center of the rectangle is positioned at the point given by the values in the *x* and *y* parameters.

c1   The number of elements in the *spec_ array* parameter

spec_ array[c1]
An array of numbers as defined by the *atype* parameter

c2   The maximum number of hits which can be returned in the *seg_ array* and *tag_ array* parameters

depth   The number of segment and tag pairs to be returned for each hit

seg_ array[c2][depth]
An array of segment identifiers. For each hit, a set of *depth* values are returned.

tag_ array[c2][depth]
An array of primitive tags. For each hit, a set of *depth* values are returned.

*num_ hits
A variable in which the number of hits in the *seg_ array* and *tag_ array* parameters is returned.

A 'hit' is an instance of a segment identifier and tag pair for which the primitives lie completely or partially within the specified aperture. Two different primitives in the same segment might have the same tag, and would therefore produce the same hit. This is counted as a single hit; the hit is only recorded once in the *seg_ array* and *tag_ array* that are returned. The *num_ hits* parameter, therefore, returns this distinct number of hits.

The tables *seg_ array* and *tag_ array* are set to the hits that are found, up to the maximum defined in the *c2* parameter. Corresponding sets of elements form the 'hit' pairs. The number returned in *num_ hits* therefore contains the number of sets of *depth* pairs set if the *c2* parameter is greater than the number of hits detected. The number of elements set in the *seg_ array* and *tag_ array* parameters is the number returned in *num_ hits* multiplied by the *depth*.

If the same value is returned in the *num_ hits* parameter as is specified in the *c2* parameter, there may be

yet more hits that cannot be returned in *seg_ array*
and *tag_ array*. If all hits are important, specify arrays
that are large enough to contain the maximum number
of hits that are expected.

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_NAMED_SEG_DOES_NOT_EXIST (I.E. NAME1)
GPIERR_NAMED_SEG_NOT_CHAINED (I.E. NAME1)
GPIERR_INVALID_SEG_ID (I.E. NAME1 or NAME2)
GPIERR_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_STROKES_DEFN_NOT_COMPLETE (at segment end)
```

GpiCorrelateSegment

```
SHORT GpiCorrelateSegment (hgpi, name, ctype,
        x, y, atype, c1, spec_array, c2, depth,
        seg_array, tag_array, num_hits)
HPS hgpi;
LONG name;
LONG ctype;
LONG x;
LONG y;
LONG atype;
LONG c1;
LONG spec_array[];
LONG c2;
LONG depth;
LONG seg_array[];
LONG tag_array[];
LONG num_hits;
```

Performs a correlate operation on the specified segment.

Data is returned for each tagged primitive that intersects the
specified aperture. The data returned for each "hit" (or correla-
tion) consists of a set of segment and tag pairs, starting with the
correlated one, then the one which called that segment, repeated
until the root segment (which was not called by another seg-
ment) is reached.

The root segment name must be non-zero.

The depth value specifies the number of sets of segment and tag
pairs to be returned for each hit. If the root segment is reached
before *depth* values, the remaining values are set to zero. If more
than *depth* values are available, only that number are returned.

The draw controls (see GpiSetDrawControl) are ignored by this
function.

If this function is followed by primitives or attributes, without first opening a segment, then the processing will be as described for GpiCloseSegment.

Parameters:

hgpi   The handle for the GPI presentation space.

name   Specifies the root segment to be correlated. It must be > 0.

ctype   The type of segments on which correlation is to be performed:-

> 0 - PICKSEL_ VISIBLE
> > Only visible and detectable segments, with nonzero identifiers, are correlated.
>
> 1 - PICKSEL_ ALL
> > All segments with nonzero identifiers are correlated, regardless of the detectability and visibility attributes of the segments.

x,y   The co-ordinates of the position of the center of the aperture in Model Space

atype   The type of aperture to be used:-

> 0 - PICKAP_ DEFAULT
> > The default; same as 1.
>
> 1 - PICKAP_ SCALED
> > Scaled pick aperture
> >
> > The *spec_ array* parameter must contain a single element, which is a uniform scaling aperture that is applied to the device's default pick aperture. In this case the binary point of the *spec_ array* parameter is considered to be between the second and third bytes; thus 65536 represents the value unity.
>
> 2 - PICKAP_ RECT
> > Rectangular aperture
> >
> > The *spec_ array* parameter must contain two Model Space values, giving the width and height (respectively) of a rectangular aperture. The center of the rectangle is positioned at the point given by the values in the *x* and *y* parameters.

c1   The number of elements in the *spec_ array* parameter

spec_ array[c1]
    An array of numbers as defined by the *atype* parameter

c2   The maximum number of hits which can be returned
    in the *seg_ array* and *tag_ array* parameters

depth The number of segment and tag pairs to be returned
    for each hit

seg_ array[c2][depth]
    An array of segment identifiers. For each hit, a set of
    depth values are returned.

tag_ array[c2][depth]
    An array of primitive tags. For each hit, a set of *depth*
    values are returned.

*num_ hits
    A variable in which the number of hits in the
    seg_ array and *tag_ array* parameters is returned.

    A 'hit' is an instance of a segment identifier and tag
    pair for which the primitives lie completely or partially
    within the specified aperture. Two different primitives
    in the same segment might have the same tag, and
    would therefore produce the same hit. This is counted
    as a single hit; the hit is only recorded once in the
    seg_ array and *tag_ array* that are returned. The
    num_ hits parameter, therefore, returns this distinct
    number of hits.

    The tables *seg_ array* and *tag_ array* are set to the hits
    that are found, up to the maximum defined in the *c2*
    parameter. Corresponding sets of elements form the
    'hit' pairs. The number returned in *num_ hits* therefore
    contains the number of sets of *depth* pairs set if the *c2*
    parameter is greater than the number of hits detected.
    The number of elements set in the *seg_ array* and
    tag_ array parameters is the number returned in
    num_ hits multiplied by the *depth*.

    If the same value is returned in the *num_ hits* parame-
    ter as is specified in the *c2* parameter, there may be
    yet more hits that cannot be returned in *seg_ array*
    and *tag_ array*. If all hits are important, specify arrays
    that are large enough to contain the maximum number
    of hits that are expected.

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_NAMED_SEG_DOES_NOT_EXIST
GPIERR_INVALID_SEG_ID (I.E. NAME)
GPIERR_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (at segment end)
GPIERR_STROKES_DEFN_NOT_COMPLETE (at segment end)
```

7.1.13.6   Bounds Data Functions

GpiResetBoundaryData

```
BOOL GpiResetBoundaryData (hgpi)
HPS hgpi;
```

Resets the boundary data to null.

This function is only necessary for draw mode boundary determination. Note that bounds data is not reset at the start of a segment.

Bounds data is automatically reset before any Draw function.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
```

GpiQueryBoundaryData

```
BOOL GpiQueryBoundaryData (hgpi, boundary)
HPS hgpi;
GRECT boundary;
```

Returns the boundary data that was set upon completion of the last boundary calculation. Boundary data is returned as the coordinates in model space.

Parameters:

hgpi   The handle for the GPI presentation space. resides.

boundary
           A rectangle structure in which the boundary data is returned.

The data contains the following fields:

xmin	lowest x value found
ymin	lowest y value found
xmax	highest x value found
ymax	highest y value found

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION

7.1.14 Segment Manipulation Functions

Segment manipulation functions fall into three classes:-

1. Those which operate on whole segments

2. Those which manipulate the contents of a segment indirectly

None of the functions described in these sections are allowed to a micro-PS.

7.1.14.1 Whole Segment Functions

GpiOpenSegment

```
BOOL GpiOpenSegment (hgpi, name)
HPS hgpi;
LONG name;
```

Opens a segment.

If the current drawing mode is store or draw-and-store (see GpiSetDrawingMode), the following occurs:-

*   If a non-zero name is given, then if a segment with the specified name does not already exist, a new stored segment is created. If one does already exist, it is re-opened in store mode, but is an error in draw-and-store mode.

*   If a name of zero is given, then a new stored segment is created, regardless of whether or not one with a zero name already exists. There can be more than one segment with a name of zero, but such segments can never subsequently be referenced by name. Once created, they will continue to exist until all segments are deleted. It is an error to attempt to open a segment zero with either the *dynamic*, or the *not*

chained segment atrributes.

If the current drawing mode is draw, a new non-stored segment is started. No check will be made against any possible stored segment names. The current attributes will be set to default values (subject to the *fast chaining* segment attribute - see below).

The initial attributes of the segment are as set by GpiSetInitial-SegmentAttrs (which see for default values). The attributes may subsequently be changed with GpiSetSegmentAttrs (except for a segment with a name of zero). It is an error to attempt to open a new segment in draw or draw-and-store mode, with the *dynamic* segment attribute.

GpiOpenSegment causes a segment bracket to be started. While the bracket is in effect, any primitive and attribute functions are considered to be part of the segment, and will be stored in it if the drawing mode is Store or Draw-and-store. The bracket will be terminated by a GpiCloseSegment. It is an error if GpiOpen-Segment is issued when a segment is already open.

The following occurs when drawing of a chained segment is started (either as it is passed across the API in draw or draw-and-store mode, or as it is found during a GpiDraw operation), unless the segment has the *fast chaining* attribute:-

- Current attributes are reset to default values

- Current model transform is reset to unity

- Current position is set to (0,0)

- The current clip area and viewing limits are reset to no clipping

- The current window/viewport transform is reset to unity

If the segment has the *fast chaining* attribute, the system may choose whether or not to perform these operations. It is the application's responsibility to ensure that either choice will produce the same results.

Parameters:

hgpi	The handle of the GPI presentation space.
name	The segment name. Negative names should not be used.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (I.E. NAME)
GPIERR_GRAPHICS_SEG_IS_CURRENT (I.E. OPEN)
GPIERR_AREA_DEFN_NOT_COMPLETE
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE
GPIERR_STROKES_DEFN_NOT_COMPLETE
```

GpiCloseSegment

```
BOOL GpiCloseSegment (hgpi)
HPS hgpi;
```

Closes the current segment.

Any subsequent primitives, not preceded by a GpiOpenSegment function, will not be stored, irrespective of the current drawing mode.

If any of the following brackets is currently open:-

• Area

• Clip area

• Strokes

then it will be aborted.

In draw or draw-and-store mode a warning will be given, but the close processing will continue. In store mode, no warning will be given. If a stored segment with one of these unended brackets is subsequently drawn, an error will be raised.

If an element bracket is open when a segment is closed, then the element bracket is first closed automatically.

If this function is followed by primitives or attributes, without first opening a segment, then the following may or may not have been reset to their default values:-

• Current attribute values

• Current model transform

• Current position

• The current clip area and viewing limits

Any such quantity may only be assumed to contain its default value if it is known either that it has not been changed from it, or that last time it was changed, it was set to its default value.

The current window/viewport transform, however, is guaranteed to be reset to unity for primitives outside segments.

Parameters:

hgpi   The handle of the GPI presentation space.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_NO_CURRENT_GRAPHICS_SEG
GPIERR_AREA_DEFN_NOT_COMPLETE (warning)
GPIERR_CLIP_AREA_DEFN_NOT_COMPLETE (warning)
GPIERR_STROKES_DEFN_NOT_COMPLETE (warning)
```

GpiDeleteSegment

```
BOOL GpiDeleteSegment (hgpi, name)
HPS hgpi;
LONG name;
```

Deletes the specified segment.

If the segment is open when it is deleted, there will be no open segment after this function.

If the segment is in the picture chain, it is removed from the chain.

Parameters:

hgpi   The handle of the GPI presentation space.

name   The name of the segment to be deleted. It must be > 0.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (I.E. NAME)
GPIERR_SEG_UNKNOWN
```

GpiDeleteSegments

```
BOOL GpiDeleteSegments (hgpi, name1, name2)
HPS hgpi;
LONG name1;
LONG name2;
```

Deletes all segments in the given name range. Note that *name1* and *name2* can have the same value; in this case, only the named segment is destroyed. If *name1* is greater than *name2* then only the segment with *name1* is destroyed.

If one of the segments deleted is the currently open segment, there will be no open segment after this function.

If any of the segments are in the picture chain, they are removed from the chain.

Parameters:

hgpi	The handle of the GPI presentation space.
name1	The first name in the range. It must be > 0.
name2	The last name in the range. It must be > 0.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (I.E. NAME1 or NAME2)
GPIERR_SEG_UNKNOWN
```

GpiQuerySegmentNames

```
LONG GpiQuerySegmentNames (hgpi, name1, name2, n, names)
HPS hgpi;
LONG name1;
LONG name2;
LONG n;
LONG names[];
```

This returns the names of all segments that exist with names in a specified name range. Non-stored segment names will not be included. If *name1* is the same as or greater than *name2* then the search will terminate after querying only the segment with *name1*.

Parameters:

hgpi	The handle of the GPI presentation space.
name1	The first name in the range. It must be > 0.
name2	The last name in the range. It must be > 0.
n	The maximum number of names to be returned in *names*.
names[n]	An array in which the required names are returned.

Returns:

```
-1 Error
>=0 Number of names returned
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (I.E. NAME1 or NAME2)
GPIERR_INVALID_ARRAY_COUNT
```

GpiSetInitialSegmentAttrs

```
BOOL GpiSetInitialSegmentAttrs (hgpi, attribute, value)
HPS hgpi;
LONG attribute;
LONG value;
```

This function sets a segment attribute which is to be assumed by
subsequent segments when they are initially created (ie when
GpiOpenSegment is issued, and the segment does not already
exist). See the section, "Segment Attributes", for an explana-
tion of segment attributes, including default settings.

Parameters:

hgpi   The handle of the GPI presentation space.

attribute
            Specifies which segment attribute is to be changed, as
            follows:-

            1 Detectability
            2 Visibility
            3 Highlighting
            6 Chained
            7 Contains prolog
            8 Dynamic
            9 Fast chaining
            10 Propagate detectability
            11 Propagate visibility

value   The required value of the attribute, as follows:-

            0 Off/no
            1 On/yes

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ATTR_CODE
GPIERR_INVALID_SEG_ATTR
```

GpiQueryInitialSegmentAttrs

```
LONG GpiQueryInitialSegmentAttrs (hgpi, attribute)
HPS hgpi;
LONG attribute;
```

This function returns an initial segment attribute.

Parameters:

hgpi   The handle of the GPI presentation space.

attribute

Specifies which initial segment attribute is to be
returned, as follows:-

1 Detectability
2 Visibility
3 Highlighting
6 Chained
7 Contains prolog
8 Dynamic
9 Fast chaining
10 Propagate detectability
11 Propagate visibility

Returns:

-1 Error
>=0 Attribute value, as follows:-
    0 Off/no
    1 On/yes

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ATTR_CODE

GpiSetSegmentAttrs

BOOL GpiSetSegmentAttrs (hgpi, name, attribute, value)
HPS hgpi;
LONG name;
LONG attribute;
LONG value;

This function sets a segment attribute for the specified segment.
The segment may be any stored segment.

If the name is that of the currently open segment

- In store mode this is valid

- In draw-and-store mode, the stored segment is updated, but
  there is no change to the immediate drawing

- In draw mode, it is invalid

Parameters:

hgpi   The handle of the GPI presentation space.

name   The name of the segment whose attribute is to be
          updated. It must be > 0.

attribute

Specifies which segment attribute is to be changed, as
follows:-

```
 1 Detectability
 2 Visibility
 3 Highlighting
 6 Chained
 7 Contains prolog
 8 Dynamic
 9 Fast chaining
10 Propagate detectability
11 Propagate visibility
```

value   The required value of the attribute, as follows:-

```
0 Off/no
1 On/yes
```

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_SEG_UNKNOWN
GPIERR_INVALID_SEG_ATTR_CODE
GPIERR_INVALID_SEG_ATTR
GPIERR_NOT_IN_STORE_MODE (CAN'T CHANGE ATTRS OF CURR SEG)
```

GpiQuerySegmentAttrs

```
LONG GpiQuerySegmentAttrs (hgpi, name, attribute)
HPS hgpi;
LONG name;
LONG attribute;
```

This function returns a segment attribute for the specified segment. The segment may be any stored segment (including the currently open one if stored).

Parameters:

hgpi   The handle of the GPI presentation space.

name   The name of the segment whose attribute is to be
          returned. It must be > 0.

attribute
          Specifies which segment attribute is to be returned, as
          follows:-

```
1 Detectability
2 Visibility
3 Highlighting
6 Chained
7 Contains prolog
8 Dynamic
9 Fast chaining
```

133

10 Propagate detectability
11 Propagate visibility

Returns:

-1 Error
>=0 Attribute value, as follows:-
　　0 Off/no
　　1 On/yes

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_SEG_UNKNOWN
GPIERR_INVALID_SEG_ATTR_CODE

GpiSetSegmentPriority

```
BOOL GpiSetSegmentPriority (hgpi, name, ref_name, order)
HPS hgpi;
LONG name;
LONG ref_name;
LONG order;
```

This function changes the order of the specified segment within the segment chain.

In the stored segment model, the application may redraw the picture by drawing the segment chain (see GpiDrawChain). This causes the segments in the chain to be processed from beginning to end, so that if segments overlap, later ones will be drawn on top (assuming a default mix mode) of earlier ones, and will therefore appear to have higher priority. Changing the position of the segment in the chain therefore has the effect of changing its priority to the end user.

Parameters:

hgpi　　The handle of the GPI presentation space.

name　　The name of the segment whose priority is to be updated. It must be $>= 0$.

ref_ name
　　　　The name of a reference segment. This is the one which the segment specified by *name* is to be positioned either immediately before or immediately after.

　　　　If *ref_ name* is *0*, then *name* will be positioned either first or last in the chain, depending upon the value of *order.*

　　　　ref_ name must be > 0.

order　　The position required for *name* relative to *ref_ name*, as follows:-

> *-lname* is to be lower priority than *ref_ name*
> (if *ref_ name* =0 then *name* is to
> be the *highest* priority segment)
> *lname* is to be higher priority than *ref_ name*
> (if *ref_ name* =0 then *name* is to
> be the *lowest* priority segment)

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_INVALID_REFSEG_ID (REF_NAME)
GPIERR_SEG_ID_UNKNOWN
GPIERR_REFSEG_ID_UNKNOWN
GPIERR_GRAPHICS_SEG_IS_CURRENT (I.E. OPEN)
GPIERR_ORDERING_PARAMETER_INVALID
GPIERR_SEG_AND_REFSEG_ARE_SAME
GPIERR_SEG_NOT_CHAINED
GPIERR_REFSEG_NOT_CHAINED
```

GpiQuerySegmentPriority

```
LONG GpiQuerySegmentPriority (hgpi, ref_name, order)
HPS hgpi;
LONG ref_name;
LONG order;
```

This function returns the identifier of the named segment which is before or after the specified segment. The segment which is before the specfied segment is considered to have a lower priority than the specified segment; the segment which is after the specfied segment is considered to have a higher priority than the specified segment.

Parameters:

hgpi   The handle of the GPI presentation space.

ref_ name
The name of the reference segment.

If *ref_ name* is *0*, then the identifier of either the first or the last segment in the chain will be returned, depending upon the value of *order*.

ref_ name must be >= 0.

order   Shows whether the segment immediately before or after *ref_ name* is required, as follows:-

-1 Return the next segment with *lower* priority than *ref_ name*
(if *ref_ name* =0 then the segment with the

> *lowest* priority is required)
> *1* Return the next segment with *higher* priority
> than *ref_ name*
> (if *ref_ name* =*0* then the segment with the
> *highest* priority is required)

Returns:

-1 Error
>=0 Name of the segment immediately before or after
ref_ name (or first or last in the chain) . *0* is
returned if there is not one

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_REFSEG_ID (REF_NAME)
GPIERR_REFSEG_ID_UNKNOWN
GPIERR_ORDERING_PARAMETER_INVALID
GPIERR_REFERENCE_SEG_NOT_CHAINED
```

7.1.14.2 Segment Content Manipulation Functions (Indirect)

Elements

A segment is constructed by means of API calls. Typically these are calls
to cause certain primitives to be drawn (eg GpiLine) or to set attributes
(eg GpiSetColor). Each such API function generates one *element* of the
segment.

The currently open segment has an *element pointer*, which points to a par-
ticular element in the segment. When a stored segment is first opened, the
element pointer is set to zero (empty segment). It is incremented each
time a call causes an element (a single API call) to be placed in the seg-
ment. When a segment is re-opened, the element pointer will be set to
zero, ie before the first element. In this position, if an element is inserted,
it will be the first element in the segment. Essentially each element is put
into the segment at the place indicated by the element pointer.

Segment zero cannot be edited.

The element pointer for a segment is not remembered if the segment is
closed and subsequently re-opened. Functions requiring an element
pointer are only valid if the currently open segment is stored, and indeed
only if the current drawing mode (see GpiSetDrawingMode) is store.
(They are not valid in draw-and-store mode.)

Associated with each element are a *type* and *description data*.

type is a long integer. For elements generated directly from calls, it is set to a system-defined value, depending upon the call. For an element generated via a data buffer (see GpiElement) the application defines the type, from a specific range of values.

description data is a variable length string. For system-defined element types this is also system-defined (description data for system-defined element types will not be provided in Presentation Manager release 1). For an element generated by GpiElement the application defines the description.

Two editing modes are provided:-

- Insert mode

  In this mode, element generating API calls will insert an element following the element indicated by the element pointer.

- Replace mode

  In this mode, element generating API calls will replace the element indicated by the element pointer. Note that it is an error to replace an element with the element pointer at zero.

Labels

A function is provided to create a *label* within a segment. A label is itself an element, which is inserted into the segment. It may subsequently be used to reference the point at which it was inserted. For example, the element pointer may be set to point to the element defined by a particular label.

Labels need not be unique within the presentation space. Indeed, they need not even be unique within a segment, although normally they will be.

The use of labels to set the element pointer may in some circumstances be faster than setting it to a particular element number.

GpiSetEditMode

```
BOOL GpiSetEditMode (hgpi, mode)
HPS hgpi;
LONG mode;
```

Sets the current editing mode for the specified presentation space.

This determines whether data is to be inserted into a segment, moving any subsequent elements further along the segment, or whether each element is to replace the current element.

The editing mode may be changed at any time, and is not an attribute of a specific segment. It only applies to the storing of

137

data within stored segments, though it is not an error to issue this function in other drawing modes. It is invalid within an element bracket. The default editing mode (set by GpiCreatePS or GpiResetPS) is *insert*.

Parameters:

hgpi	The handle of the GPI presentation space.
mode	The mode, as follows:-

> *1* Insert mode
> *2* Replace mode

Returns:

> *0* Error
> *1* OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_EDIT_MODE
```

GpiQueryEditMode

```
LONG GpiQueryEditMode (hgpi)
HPS hgpi;
```

Returns the current editing mode (see GpiSetEditMode).

This function may be issued in any drawing mode.

Parameters:

hgpi	The handle of the GPI presentation space.

Returns:

> *0* Error
> *>0* Current editing mode

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
```

GpiSetElementPointer

```
BOOL GpiSetElementPointer (hgpi, element)
HPS hgpi;
LONG element;
```

Sets the element pointer, within the current segment, to the element number specified.

If the value specified is negative, the element pointer is set to 0. If the value specified is greater than the number of elements in the segment, it is set to the last element.

This function is only valid when the drawing function mode is

set to Store (not Draw-and-store), and a segment bracket is currently in progress. It is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

element   The element number required.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_ELEMENT_NUMBER
GPIERR_ELEMENT_CONTEXT_ERROR
GPIERR_NOT_IN_STORE_MODE
GPIERR_NO_CURRENT_GRAPHICS_SEG
```

GpiQueryElementPointer

```
LONG GpiQueryElementPointer (hgpi)
HPS hgpi;
```

Returns the current element pointer (see GpiSetElementPointer).

This function is only valid when the drawing function mode is set to Store (not Draw-and-store), and a segment bracket is currently in progress.

Parameters:

hgpi   The handle of the GPI presentation space.

Returns:

-1 Error
>=0 Current element pointer

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_ELEMENT_NUMBER
GPIERR_NOT_IN_STORE_MODE
GPIERR_NO_CURRENT_GRAPHICS_SEG
```

GpiOffsetElementPointer

```
BOOL GpiOffsetElementPointer (hgpi, offset)
HPS hgpi;
LONG offset;
```

Sets the element pointer, within the current segment, to the current value plus the specified offset.

If the resulting value is negative, the element pointer is set to 0.

139

If the resulting value is greater than the number of elements in the segment, it is set to the last element.

This function is only valid when the drawing function mode is set to Store (not Draw-and-store), and a segment bracket is currently in progress. It is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

offset   The offset which is to be added to the element pointer.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_NOT_IN_STORE_MODE
GPIERR_NO_CURRENT_GRAPHICS_SEG
GPIERR_ELEMENT_CONTEXT_ERROR
```

GpiDeleteElement

```
BOOL GpiDeleteElement (hgpi)
HPS hgpi;
```

Deletes the element indicated by the element pointer. The element pointer is set to the element immediately preceding the deleted element.

If the element pointer has a value of 0 (points logically before the first element), nothing is deleted and the element pointer is not changed.

This function is only valid when the drawing function mode is set to Store (not Draw-and-store), and a segment bracket is currently in progress. It is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_NOT_IN_STORE_MODE
GPIERR_NO_CURRENT_GRAPHICS_SEG
GPIERR_ELEMENT_CONTEXT_ERROR
```

GpiDeleteElementRange

```
BOOL GpiDeleteElementRange (hgpi, element1, element2)
HPS hgpi;
LONG element1;
LONG element2;
```

Deletes all elements between and including the elements indicated by the specified element numbers.

If either element number is outside the range of the current segment, it is set to the nearest valid value.

At the conclusion of this function, the element pointer is set to the element immediately preceding the deleted elements.

This function is only valid when the drawing function mode is set to Store (not Draw-and-store), and a segment bracket is currently in progress. It is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

element1,element2
            The numbers of the first and last elements to be deleted.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_ELEMENT_NUMBER
GPIERR_NOT_IN_STORE_MODE
GPIERR_NO_CURRENT_GRAPHICS_SEG
GPIERR_ELEMENT_CONTEXT_ERROR
```

GpiLabel

```
BOOL GpiLabel (hgpi, label)
HPS hgpi;
LONG label;
```

Generates an element containing the specified label. This has no effect unless a stored segment is being constructed.

This function is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

label   The required label. No check is made on the value of *label*.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_ELEMENT_CONTEXT_ERROR

GpiSetElementPointerAtLabel

BOOL GpiSetElementPointerAtLabel (hgpi, label)
HPS hgpi;
LONG label;

Sets the element pointer, within the current segment, to the element containing the specified label.

The search will start from the next element beyond the one which the element pointer is currently pointing to. If no occurrence of the specified label is found between there and the end of the segment, an error will be generated and the element pointer left unchanged.

This function is only valid when the drawing function mode is set to Store (not Draw-and-store), and a segment bracket is currently in progress. It is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

label   The label required.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_LABEL_NOT_FOUND
GPIERR_NOT_IN_STORE_MODE
GPIERR_NO_CURRENT_GRAPHICS_SEG
GPIERR_ELEMENT_CONTEXT_ERROR

GpiDeleteElementsBetweenLabels

BOOL GpiDeleteElementsBetweenLabels (hgpi, label1, label2)
HPS hgpi;
LONG label1;
LONG label2;

Deletes all elements between but not including the elements found to contain the indicated labels.

The search for the elements is conducted, separately, in the same

way as described for GpiSetElementPointerAtLabel. If either label cannot be found between the current element pointer location and the end of the segment, an error is generated and no deletion occurs.

At the conclusion of this function, the element pointer is set to the element immediately preceding the deleted elements.

This function is only valid when the drawing function mode is set to Store (not Draw-and-store), and a segment bracket is currently in progress. It is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

label1,label2
          The labels marking the bounds of the elements to be deleted.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_LABEL_NOT_FOUND (LABEL1)
GPIERR_LABEL2_NOT_FOUND (LABEL2)
GPIERR_NOT_IN_STORE_MODE
GPIERR_NO_CURRENT_GRAPHICS_SEG
GPIERR_ELEMENT_CONTEXT_ERROR
```

GpiQueryElementType

```
LONG GpiQueryElementType (hgpi, type, desc_length, desc)
HPS hgpi;
LONG *type;
LONG *desc_length;
LPBUF desc;
```

Returns information about the element which the element pointer currently points to.

This function is only valid when the drawing function mode is set to Store (not Draw-and-store), and a segment bracket is currently in progress. It is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

*type   A variable in which the type of the element is returned. The type may be system-defined or application-defined (see GpiElement)

*desc_length
    A variable which contains the length of data in the buffer pointed to by *desc.* On return, it is updated to the number of bytes actually stored.

desc   A variable in which the description data for the element is returned. The description may be system-defined or application-defined (see GpiElement) *Note:* system-defined element data is not supported in the current version of Presentation Manager.

Returns:

```
-1 Error
>=0 Size of the data required to hold the
    element content.  This may be used for a
    subsequent GpiQueryElement function.
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_CURRENT_ELEMENT_DOES_NOT_EXIST
GPIERR_NOT_IN_STORE_MODE
GPIERR_INVALID_LENGTH
GPIERR_ELEMENT_CONTEXT_ERROR
```

GpiQueryElement

```
LONG GpiQueryElement (hgpi, start, length, buffer)
HPS hgpi;
LONG start;
LONG length;
LPBUF buffer;
```

Returns the element content (or part of the element content) for the element which the element pointer currently points to.

This function is only valid when the drawing function mode is set to Store (not Draw-and-store), and a segment bracket is currently in progress. It is invalid within an element bracket.

Parameters:

hgpi   The handle of the GPI presentation space.

start   The starting byte offset within the content.

length   The maximum length of data which may be returned.

buffer   An area of *length* bytes in which the element content data is to be returned.

Returns:

```
-1 Error
>=0 Actual number of bytes returned
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_START
GPIERR_CURRENT_ELEMENT_DOES_NOT_EXIST
GPIERR_NOT_IN_STORE_MODE
GPIERR_INVALID_LENGTH
GPIERR_ELEMENT_CONTEXT_ERROR
```

GpiElement

```
SHORT GpiElement (hgpi, type, desc, length, buffer)
HPS hgpi;
LONG type;
LPSZ desc;
LONG length;
LPBUF buffer;
```

Specifies a complete element which is to be stored in the current segment (in Store or Draw-and-store mode). The element will be drawn in Draw or Draw-and-store mode.

It is an error if the element data contains any begin or end element orders. Similarly, GpiElement is invalid within an element bracket.

Note that no co-ordinate conversion is performed by this function. The application must ensure that the co-ordinates within the element are in the correct format for the presentation space.

Parameters:

hgpi	The handle of the GPI presentation space.
type	The type which is to be associated with the element.
desc	A variable length character string which is recorded with the type
length	The length of content data for the element.
	This must not be greater than 63K.
buffer	A pointer to a buffer of *length* bytes, which contains the element content data. The format of the data is TBD.

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_ELEMENT_TYPE
GPIERR_INVALID_LENGTH
GPIERR_DATA_TOO_LONG
GPIERR_INVALID_ORDER
GPIERR_ELEMENT_CONTEXT_ERROR
```

GpiBeginElement

```
BOOL GpiBeginElement (hgpi, type, desc)
HPS hgpi;
LONG type;
LPSZ desc;
```

Specifies the start of an element, which will be stored in the current segment (in Store or Draw-and-store mode). The element will be drawn in Draw or Draw-and-store mode.

The primitives and attributes which are contents of the element will be passed on subsequent API functions. GpiElement, which itself generates a complete element is not allowed within an element bracket. The element extends up to the next GpiEndElement function (or GpiCloseSegment, which causes an implicit GpiEndElement to be generated).

Elements may not be nested.

Parameters:

hgpi   The handle of the GPI presentation space.

type   The type which is to be associated with the element.

desc   A variable length character string which is recorded with the type

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_ATTEMPT_TO_START_SECOND_ELEMENT
GPIERR_INVALID_ELEMENT_TYPE
GPIERR_ELEMENT_CONTEXT_ERROR
```

GpiEndElement

```
BOOL GpiEndElement (hgpi)
HPS hgpi;
```

Terminates an element, which had been started by GpiBeginElement.

Parameters:

hgpi   The handle of the GPI presentation space.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_END_ELEMENT_IGNORED
GPIERR_ELEMENT_CONTEXT_ERROR
```

7.1.15   Transform Functions

The four co-ordinate spaces used by GPI were introduced in the section "Co-ordinate Spaces". This section defines the functions which specify the relationships between these spaces.

7.1.15.1   Co-ordinate Spaces

Applications typically work in the following kinds of units:

1. Device units

    - Actual pels

    - A simple multiple of pels, for extra granularity with scaling transforms

2. Metrics

    Fractions of an inch or of a millimetre or of a printer's point

3. Fractions of the screen

    This is convenient for constructing simple graphics which will look similar on a range of devices

4. Application convenient units

    Anything appropriate to the application, such as grid reference units, or world-oriented metrics.

A Presentation Manager application may, if it wishes, exercise direct control over the co-ordinate spaces it uses. Alternatively, it may make use of certain defaults to give some of the simpler options.

Starting from application co-ordinates, the drawing process must eventually generate device co-ordinates, and it will usually be efficient for it to make the transition from application co-ordinates to device co-ordinates in a single step. Notionally, however, there are additional intermediate co-ordinate spaces. In the general case these are required to facilitate various functions which are described below. It must be emphasised, however, that the defaulting rules mean that applications need not be directly concerned with any spaces which they do not wish to control explicitly.

The levels of co-ordinate spaces are as follows:

```
                              World Co-ordinates
            |                                                    |
            |                                                    |
    ------+------                              ------+------
            |    |                                          |    |
            |    |                                          |    |
            |    |       Segment / Model Transform          |    |
            |    |                                          |    |
            |    V                                          |    V

            |   +--------+                                 |   +--------+
            |   |Window-1|          Model                  |   |Window-2|
            |   +------%-+                                 |   +------%-+
    ------+------       |           Space          ------+------       |
            |    |                                          |    |
            |    |                                          |    |
                 |                                               |
                 |        Window / Viewport Transform            |
                 |        and Default Viewing Transform          |
                 |                                    +----------------+
                 |                                    |              |
         +-------|-------------------------|---------+              |
         | +-----|-------------------+     |         |              |
         | |     V                   |     |         |              |
         | |  +---*------+           |     V         |              |
    Page | |  |Viewport-1|       +--+---*---+        |              |
         | |  |          |       |Viewport-2|        |              |
(pels,metrics, |  +----------+       +--+-------+        |              |
 isotropic)  | |                      |              |              |
         | |  |Page window           |              |              |
         | |  +----------%-------+    |              |              |
         | |                    |    |              |              |
         +-------------------------|---------------+              |

                                 |
              Device Transform   |
                                 |
                   |             V
              +---+--------*------+
              |   |    Page viewport|
              |   |                 |
              |   >------+          |
              |   |      |          |
 Device Space |   |      |          |
  (pels)   ------+---+--%--*----------+----------
              |   |      |          |
              |   |      |          |
              +---+--|------------+
                  |   |
```

```
                                    |
                                    |
  Windowing System Translation|
                                    |
                                    |
                    +------|-----------------+
                    |      V                  |
                    |    +*----+              |
                    |    |     |Media         |
                    |    |     |Window        |
   Media Space      |    |     |              |
   (pels)           |    |     |              |
                    |    +-----+              |
                    +------------------------+
```
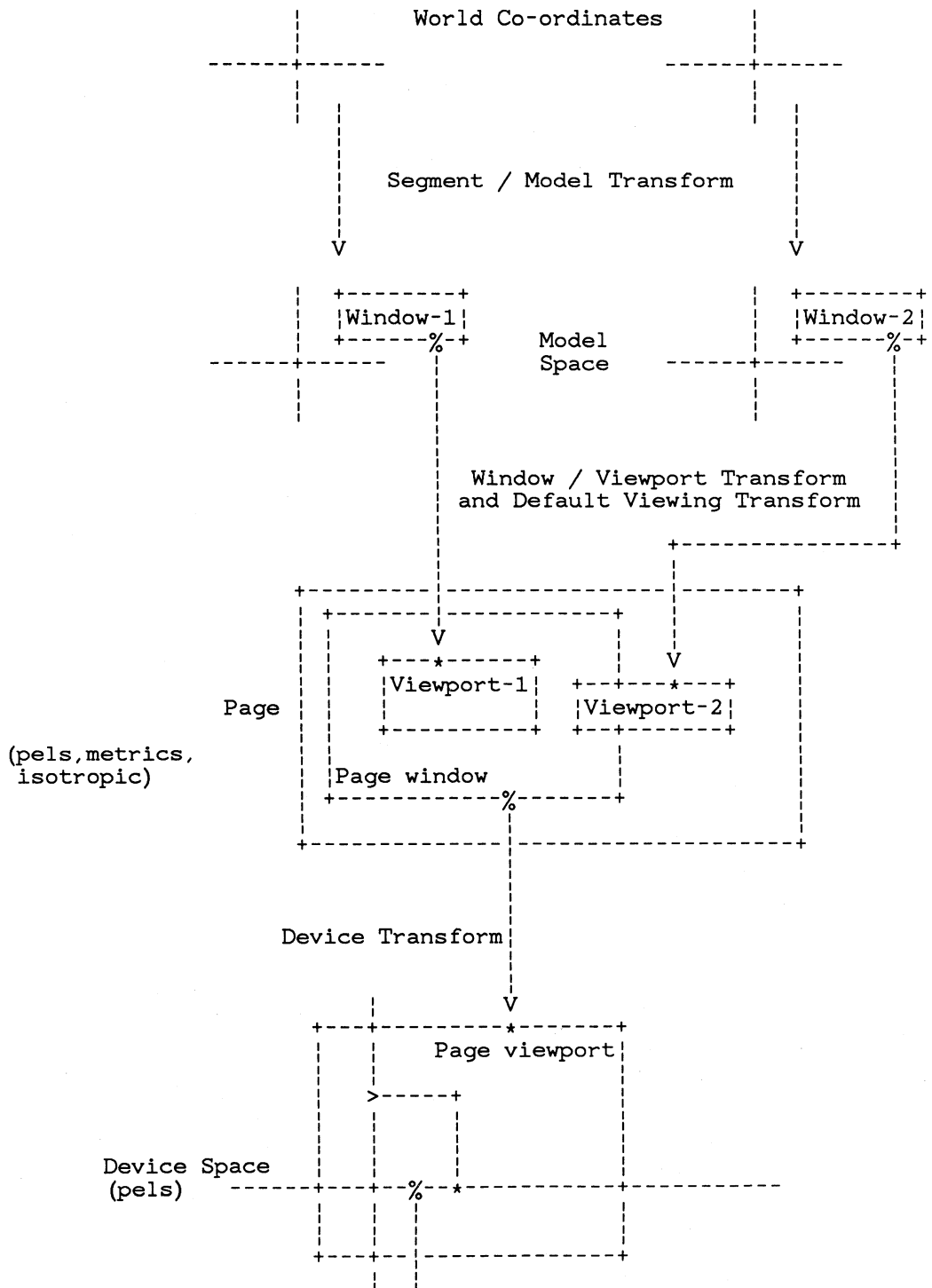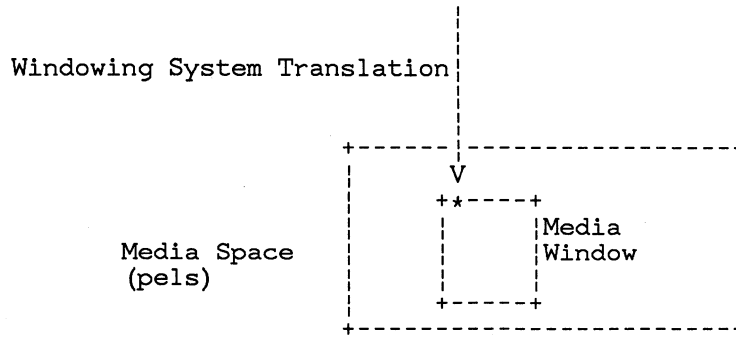
Figure 7.1 Presentation Manager Pipeline

1. World Co-ordinate Space

   These are application convenient units. They are the units which are
   used at the API for primitives such as lines, arcs etc. In store mode,
   this is the space in which primitive co-ordinates are stored.

2. Model Space

   Segments and primitives may optionally be subjected to segment and
   model transforms. These are used to construct the picture. For exam-
   ple, there may be a single segment for drawing a wheel of a locomotive,
   which is called several times, once for each wheel, using a different
   transform to position it at the correct place - possibly also to scale it.

   Model Space is equivalent to World Co-ordinate Space if no segment or
   modelling transforms are used.

3. Page

   The page is where the picture is assembled. It may be that the picture
   is composed of more than one subpicture. For example, it may be con-
   venient to construct a business graph as one subpicture containing the
   axes and lines, and another containing the legend. The final picture,
   which may include several subpictures, is assembled in the Page.

   The size of the Page may be defined in various units. See GpiCreatePS,
   and also the section, "Defaults and Examples".

   A window on the Page (the Page Window) defines the maximum area
   of the page which may be visible at any time. It also, in conjunction
   with the Page Viewport in Device Space, defines the Device Transform.

4. Device Space

   It will be seen that an application can deal directly, right from the top,
   in device co-ordinates if it wishes. Some applications, however, will
   want to construct pictures in device independent spaces. Device Space
   allows the Page above it to be device independent if the application so
   wishes. Device Space itself is defined in device units.

5. Media Space

The picture, or a portion of it, is finally displayed in the window on the screen (or the paper on a printer, etc). A final transform, which only supports translation, maps the total picture in Device Space into the screen window or printer page etc. On a display, this will be used to ensure, for example, that when the window is moved, the visible contents will not change. This transform is set only by the windowing system.

Functions are provided to convert a co-ordinate value between any one space and another.

7.1.15.2 Transforms

Between each of the above co-ordinate spaces there is a transform. These are as follows (see):-

1. Segment and Model transform

These convert from World Co-ordinates to Model Space. They are typically used during the construction of a picture, for example to scale up one construct in a picture. A segment transform applies to a whole segment; a model transform can apply to a group of primitives within a segment. For stored segments, either transform may be changed after the segment has been constructed.

One form of model transform is the instance transform which is specified when a segment is called.

These transforms support rotation.

2. Viewing Transforms

There are two componenets of the transform from Model Space to the Page, as follows:-

- Window/Viewport transform

This provides a transform from Model Space to the Page, which may be used for one or more segments. It may be used to provide a general transformation (not a rotation) to a part, or all of the picture. It may not be changed within a segment.

For stored segments, the window/viewport transform is fixed at the time the segment is created, and may not subsequently be changed without re-creating the segment.

- Default Viewing Transform

This transform, which is initialised to unity, applies after the window-viewport transform. There is a single default viewing transform, which applies across the entire picture. It may be thought of as an override to the viewport positions and sizes, and its purpose is to allow scrolling or zooming of the whole picture.

3. Device Transform

   This specifies how the Page Window is mapped into the Page Viewport in Device Space. It is a convenient way of changing all of the co-ordinates of a picture (as, for example, where all co-ordinates represent a given metric on the screen). This allows a device-independent picture to be constructed in the Page.

   Any non-squareness in the device co-ordinates (pels on a raster device) is also allowed for here.

4. Windowing System Translation

   This takes the notional picture in device space and maps it to the physical device. Its normal function is to allow for the positioning of the window on the screen. This transform is set only by the windowing system.

7.1.15.2.1 Transform Range and Precision

Internally, Presentation Manager will convert all transforms to a matrix form, as follows:-

```
( a   b   c )
( d   e   f )
( 0   0   1 )
```

so that a point with co-ordinates (x,y) is transformed to the point

```
(a*x + b*y + c, d*x + e*y + f)
```

All of the relevant transforms (model, viewing, device, etc) will be concatenated together. Intermediate results are held to the same range and precision as is each constituent transform. This is as follows:-

- For the scaling/rotation elements (a,b,d,e): 32-bit signed values, with a notional binary point between the second and third bytes (ie 1.0 is represented as 65536).

- For the translation elements (c,f) 32-bit signed integers.

In order to avoid overflow, the application should ensure that it uses values which will not cause these ranges to be exceeded, in whatever order the concatenation is performed.

7.1.15.3 Clipping

Clipping may take place logically at various points in the pipeline:

1. Clip area

   This is a shape defined by primitives. The shape may either be rectangular (in which case GpiSetViewingLimits may be used to set it), or

more general, in which case GpiBeginClipArea is used to introduce the primitives required to define the shape.

The default is the whole of space.

2. Graphics field

   This is a rectangle specified in the Page. The default is the whole Page.

3. Media window

   Data is clipped to the client area of the (media) screen window.

Clipping will take place to the intersection of all three of the above, transformed to the same space. By default, only the client area of the media window, and the page size, will be relevant.

In addition to the above, there is also clipping introduced via the clipping region. This takes place logically in device space, and may be to any irregular (even disjoint) shape.

7.1.15.4 Defaults and Examples

Before accessing Gpi functions, a presentation space must be created (Gpi-CreatePS). This function includes as parameters the specification of a Page, including its units, and its width and height.

Units are one of the following:-

- Device co-ordinates (pels)
- Metrics (various options, eg 0.1 mm)
- Arbitrary

The origin of the Page co-ordinate system is at the bottom left.

Creating a Page (of width w and height h) also sets the following:-

- A Page Window the same size (and units) as the page.
- A Window in Model Space, of (bottom left to top right):
  - (Page units = pels): from (0,0) to (w-1,h-1).
  - (Page units = metrics): from (0,0) to (w-1,h-1).
  - (Page units = arbitrary): from (0,0) to (w-1,h-1).
- A Graphics Field (clipping limit) of the same size (and units) as the Page.

Creating (or defaulting) a Graphics Field also sets the following:-

- A Viewport of the same size as the Graphics Field.

The Viewport may be changed subsequently.

The size and units of the Page, and the size and origin of the Page Window, are part of the presentation space. The Page Viewport is logically not part of the presentation space. When a presentation space is associated with a new Device Context, the Page Viewport (in pels) is computed, taking into account the pel spacing on the new device, from the following rules:-

- Page units = pels

  The Page Viewport is the same size as the Page Window, with the bottom left corner of the Page Window mapping to the origin of Device Space

- Page units = metrics

  The size of the Page Viewport gives the correct physical size as designated by the Page Window (depending upon the physical pel spacing), with the bottom left corner of the Page Window mapping to the origin of Device Space

- Page units = arbitrary

  For any device, there is a default size. In the case of a screen, this is the maximised window size. For a plotter, it is the accessible size of the paper. The Page Viewport is constructed such that Page co-ordinates will give equal x and y spacing. The bottom left corner of the Page Window maps to the origin of Device Space.

Examples

1. To use only pel co-ordinates:-
   - Specify the Page size in pels.
   - Viewports may also be laid out in pels. Each time a new viewport is defined, the window must also be redefined with the same size as the viewport, but with arbitrary origin.

2. To use only metric co-ordinates:-
   - Specify the Page size in the appropriate metrics.
   - Viewports may also be laid out in the same metrics. Each time a new viewport is defined, the window must also be redefined with the same size as the viewport, but with arbitrary origin.

3. To adjust the size of the picture depending upon the size of the client area on the screen (eg a clock):-
   - Specify a Page size in arbitrary units.

- Respecify the Page Viewport whenever the client area changes.

4. To scroll the picture in Page units

   - Change the default viewing transform, or move the page window. (The graphics field clipping boundary may affect the result if the latter method is used.)

5. To scroll the picture in page units, using a combination of BitBlt and redraw (assuming for simplicity that no model or viewing transforms are in force):-

   - Identify the amount to be scrolled in page units (DeltaP)

   - Issue GpiBitBlt, quoting the source and destination rectangles separated by DeltaP.

   - Set up a clipping region for the part of the window to be healed by redrawing. Use GpiExcludeClipRegion, quoting the rectangle that was used for the destination on GpiBitBlt.

   - Put in a translation component of DeltaP to the default view transform (GpiSetDefaultView).

   - Move the pattern origin by DeltaP.

   - Redraw the picture (possibly restrict the primitives passed to those which the application knows may contribute to the new part.

7.1.15.5 Modelling Transform Functions

GpiSetSegmentOrigin

```
BOOL GpiSetSegmentOrigin (hgpi, name, x, y)
HPS hgpi;
LONG name;
LONG x;
LONG y;
```

This sets the origin of the specified segment in world co-ordinates. This provides a reference point about which model transformations (see GpiSetModelTransform), segment transformations (see GpiSetSegmentTransform), and instance transformations (see GpiCallSegment), will be performed.

Primitives outside segments always have an effective origin of (0,0). This cannot be changed.

Parameters:

---

hgpi   The handle of the Gpi presentation space.

name   The name of the segment. It must be > 0.

x,y   The co-ordinates of the segment origin.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_SEG_UNKNOWN
```

GpiQuerySegmentOrigin

```
BOOL GpiQuerySegmentOrigin (hgpi, name, x, y)
HPS hgpi;
LONG name;
LONG *x;
LONG *y;
```

This returns the position of the segment origin of the identified segment in world co-ordinates.

Parameters:

hgpi   The handle of the Gpi presentation space.

name   The name of the segment. It must be > 0.

*x,*y   Variables which are set to the x and y co-ordinates of the segment origin.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_SEG_UNKNOWN
```

GpiSetSegmentTransform

```
BOOL GpiSetSegmentTransform (hgpi, name, sx, sy,
        hx, hy, rx, ry, dx, dy, type)
HPS hgpi;
LONG name;
LONG sx;
LONG sy;
LONG hx;
LONG hy;
LONG rx;
LONG ry;
LONG dx;
LONG dy;
LONG type;
```

Sets the two-dimensional segment transform which is to apply to all of the primitives in the specified segment.

This performs the same function as GpiSetSegmentTransform-Matrix, but specifies the transform as scale, shear, rotation and displacement components, rather than as a direct matrix.

The parameters specified are effectively combined into a single transformation matrix in the order scale, shear, rotation, displacement. This matrix is then used to update the existing segment transformation, depending upon the value of *type*.

If scaling values greater than unity are given, care must be taken that the combined effect of this and any other relevant transforms do not exceed fixed-point implementation limits. See the section, "Transform Range and Precision".

Segment transformations do not apply to primitives outside segments.

Parameters:

hgpi	The handle for the GPI presentation space.
name	The name of the segment. It must be > 0.
sx,sy	A scale transformation in terms of an x-axis scaling (*sx*) and a y-axis scaling (*sy*). The segment origin is used as a reference point; the axes that are used to scale are parallel to the x and y axes, and pass through the segment origin. A scale factor of between 0 and 1 shrinks primitives; a scale factor greater than 1 stretches primitives. A negative scale factor reflects primitives about the other axis.
	The values are each represented as signed 32-bit values, with the low-order 16 bits taken to be to the right of the binary point. Specifying scale factors of 1 and 1 (actually, in this representation, 65536 and 65536) does not perform any scaling.
hx,hy	A shear transformation in terms of the displacements which a point on the y-axis makes after shearing. The axes used for shearing are parallel to the x and y axes, but pass through the current segment origin. Note that primitives below the x axis are sheared in the opposite direction from those above the x axis. Points on the x axis itself are not moved. $hx = a$ and $hy = b$ produce an identical effect to $hx = -a$ and $hy = -b$.
	Specifying $hx = 0$ and $hy = 1$ does not perform any shearing. Specifying $hy = 0$ is invalid, because it would produce an infinite shear.

rx,ry
A rotation transformation in terms of the displacements which a point on the x-axis makes after rotating. The axes used for rotating are parallel to the x and y axes, but pass through the current segment origin.

Specifying $rx = 1$ and $ry = 0$ does not perform any rotation.

Because two zero values would be ambiguous, specifying $rx = 0$ and $ry = 0$ is taken as equivalent to $rx = 1$ and $ry = 0$ (no rotation).

dx,dy
Specify a displacement of dx parallel to the x axis, and dy parallel to the y axis. This transformation does not use the segment origin.

Specifying $dx = 0$ and $dy = 0$ does not perform any translation.

type
Specifies how the specified transform should be combined with/replace the existing segment transform. The new segment transform is computed, and the result stored back in the segment, replacing the existing value. The stored segment transform is always additive with respect to any segment, model, and instance transforms in calling segments.

0 - New/replace
The existing segment transform is discarded and replaced by the combined effect of the specified components.

1 - Additive
The combined effect of the specified components is added to the effect of the existing segment transform, in the order (1) existing transform, (2) new transform. This option is most useful for incremental updates to transforms.

2 - Preemptive
The combined effect of the specified components is added to the effect of the existing segment transform, in the order (1) new transform, (2) existing transform.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION

```
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_SEG_UNKNOWN
GPIERR_INVALID_TRANSFORM_TYPE
GPIERR_INVALID_TRANSFORM_PARAMETER
```

GpiSetSegmentTransformMatrix

```
BOOL GpiSetSegmentTransformMatrix (hgpi, name, n,
        array, type)
HPS hgpi;
LONG name;
LONG n;
LONG array[];
LONG type;
```

Sets the two-dimensional segment transform which is to apply to all of the primitives in the specified segment.

This performs the same function as GpiSetSegmentTransform, but specifies the transform as a matrix rather than scale, shear, rotation and displacement components.

The matrix is used to update the segment transformation, depending upon the value of *type*.

The transform is specified as a one-dimensional array of n elements, being the first n elements of a 3-row by 3-column matrix ordered by rows. The order of the elements is as follows:-

```
    Matrix                      Array

( a  b  c )
( d  e  f )           (a,b,c,d,e,f,0,0,1)
( 0  0  1 )
```

The last row, if specified, must be (0,0,1). The transform acts on the co-ordinates of the primitives in a segment, so that a point with co-ordinates (x,y) is transformed to the point

```
(a*x + b*y + c, d*x + e*y + f)
```

The initial value of the segment transform is the identity matrix, as shown below:-

```
    Matrix                      Array

( 1  0  0 )
( 0  1  0 )           (1,0,0,0,1,0,0,0,1)
( 0  0  1 )
```

If scaling values greater than unity are given, care must be taken that the combined effect of this and any other relevant transforms do not exceed fixed-point implementation limits. See the section, "Transform Range and Precision".

Segment transformations do not apply to primitives outside segments.

Parameters:

hgpi   The handle for the GPI presentation space.

name   The name of the segment. It must be > 0.

n   The number of elements supplied in *array*. If *n* is less than *9*, the elements omitted default to the corresponding elements of the identity matrix (see above). Specifying $n = 0$ means that the identity matrix is used. Specifying *6* elements means that the last row is assumed to be (0,0,1).

array[n] The elements of the transformation matrix, in row order.

         The first, second, fourth, and fifth elements (a, b, d and e in the example above) are specified with an assumed binary point between the second and third bytes. Thus a value of 1.0 is represented as 65536. Other elements are normal signed integers.

         The seventh, eight, and ninth elements, if specified, must be 0, 0, and 1.

type   Specifies how the specified transform should be combined with/replace the existing segment transform. The new segment transform is computed, and the result stored back in the segment, replacing the existing value. The stored segment transform is always additive with respect to any segment, model, and instance transforms in calling segments.

         0 - New/replace
                 The existing segment transform is discarded and replaced by the combined effect of the specified components.

         1 - Additive
                 The combined effect of the specified components is added to the effect of the existing segment transform, in the order (1) existing transform, (2) new transform. This option is most useful for incremental updates to transforms.

         2 - Preemptive
                 The combined effect of the specified components is added to the effect of the existing segment transform, in the order (1) new transform, (2) existing transform.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_SEG_UNKNOWN
GPIERR_INVALID_TRANSFORM_TYPE
GPIERR_INVALID_TRANSFORM_PARAMETER
```

GpiQuerySegmentTransformMatrix

```
BOOL GpiQuerySegmentTransformMatrix (hgpi, name, n, array)
HPS hgpi;
LONG name;
LONG n;
LONG array[];
```

Returns the segment transform of the specified segment. See GpiSetSegmentTransformMatrix.

Parameters:

hgpi	The handle for the GPI presentation space.
name	The name of the segment. It must be > 0.
n	The number of elements supplied in *array*. It must be in the range 0 through 9.
array[n]	The array into which the elements of the segment transform matrix will be returned.

The first, second, fourth, and fifth elements will be returned with an assumed binary point between the second and third bytes. Thus a value of 1.0 is represented as 65536. Other elements are normal signed integers.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_ARRAY_COUNT
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_SEG_UNKNOWN
```

GpiSetModelTransform

```
BOOL GpiSetModelTransform (hgpi, sx, sy, hx, hy,
        rx, ry, dx, dy, type)
HPS hgpi;
LONG sx;
LONG sy;
LONG hx;
LONG hy;
LONG rx;
```

```
LONG ry;
LONG dx;
LONG dy;
LONG type;
```

Sets the two-dimensional model transform which is to apply to subsequent primitives in this segment.

The parameters specified are effectively combined into a single transformation matrix in the order scale, shear, rotation, displacement. This matrix is then used to update the previous current model transformation, depending upon the value of *type*.

If scaling values greater than unity are given, care must be taken that the combined effect of this and any other relevant transforms do not exceed fixed-point implementation limits. See the section, "Transform Range and Precision".

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The default model transform is the unity transform, with zero translation.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi	The handle for the GPI presentation space.
sx,sy	A scale transformation in terms of an x-axis scaling (*sx*) and a y-axis scaling (*sy*). The segment origin is used as a reference point; the axes that are used to scale are parallel to the x and y axes, and pass through the segment origin. A scale factor of between 0 and 1 shrinks primitives; a scale factor greater than 1 stretches primitives. A negative scale factor reflects primitives about the other axis.
	The values are each represented as signed 32-bit values, with the low-order 16 bits taken to be to the right of the binary point. Specifying scale factors of 1 and 1 (actually, in this representation, 65536 and 65536) does not perform any scaling.
hx,hy	A shear transformation in terms of the displacements which a point on the y-axis makes after shearing. The axes used for shearing are parallel to the x and y axes, but pass through the current segment origin. Note that primitives below the x axis are sheared in the opposite direction from those above the x axis. Points on the x axis itself are not moved. $hx = a$ and $hy = b$ produce an identical effect to $hx = -a$ and $hy = -b$.

Specifying $hx = 0$ and $hy = 1$ does not perform any shearing. Specifying $hy = 0$ is invalid, because it would produce an infinite shear.

rx,ry   A rotation transformation in terms of the displacements which a point on the x-axis makes after rotating. The axes used for rotating are parallel to the x and y axes, but pass through the current segment origin.

Specifying $rx = 1$ and $ry = 0$ does not perform any rotation.

Because two zero values would be ambiguous, specifying $rx = 0$ and $ry = 0$ is taken as equivalent to $rx = 1$ and $ry = 0$ (no rotation).

dx,dy   Specify a displacement of dx parallel to the x axis, and dy parallel to the y axis. This transformation does not use the segment origin.

Specifying $dx = 0$ and $dy = 0$ does not perform any translation.

type   Specifies how the specified transformation should be used to modify the existing current model transformation (the existing transformation is the concatenation, in the current call context, of the instance, segment and model transformations, from the root segment downwards):-

0 - New/replace
> The previous model transform is discarded and replaced by the combined effect of the specified components.

1 - Additive
> The combined effect of the specified components is added to the effect of the existing model transform, in the order (1) existing transform, (2) new transform. This option is most useful for incremental updates to transforms.

2 - Preemptive
> The combined effect of the specified components is added to the effect of the existing model transform, in the order (1) new transform, (2) existing transform.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_TRANSFORM_TYPE
GPIERR_INVALID_TRANSFORM_PARAMETER
```

GpiCallSegment

```
SHORT GpiCallSegment (hgpi, name, sx, sy, hx, hy,
        rx, ry, dx, dy, type)
HPS hgpi;
LONG name;
LONG sx;
LONG sy;
LONG hx;
LONG hy;
LONG rx;
LONG ry;
LONG dx;
LONG dy;
LONG type;
```

Calls the specified segment.

The transformation specified by the parameters is set before calling the segment, to allow its scale, shear, rotation, and position to be specified. This transform only applies to the called segment, and is reset on return, to the transform in operation before the call was made. If the *type* parameter specifies the additive or preemptive option, the transform is combined with the previous model transform (if any) in force at the time.

The elements of the transformation are processed in the order scale, shear, rotation, and displacement.

If scaling values greater than unity are given, care must be taken that the combined effect of this and any other relevant transforms do not exceed fixed-point implementation limits. See the section, "Transform Range and Precision".

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
name	The name of the segment that is to be called. It must be > 0.
sx,sy	A scale transformation in terms of an x-axis scaling (*sx*) and a y-axis scaling (*sy*). The segment origin is used as a reference point; the axes that are used to scale are parallel to the x and y axes, and pass through the segment origin. A scale factor of between 0 and 1 shrinks primitives; a scale factor greater than 1

stretches primitives. A negative scale factor reflects primitives about the other axis.

The values are each represented as signed 32-bit values, with the low-order 16 bits taken to be to the right of the binary point. Specifying scale factors of 1 and 1 (actually, in this representation, 65536 and 65536) does not perform any scaling.

hx,hy   A shear transformation in terms of the displacements which a point on the y-axis makes after shearing. The axes used for shearing are parallel to the x and y axes, but pass through the current segment origin. Note that primitives below the x axis are sheared in the opposite direction from those above the x axis. Points on the x axis itself are not moved. $hx = a$ and $hy = b$ produce an identical effect to $hx = -a$ and $hy = -b$.

Specifying $hx = 0$ and $hy = 1$ does not perform any shearing. Specifying $hy = 0$ is invalid, because it would produce an infinite shear.

rx,ry   A rotation transformation in terms of the displacements which a point on the x-axis makes after rotating. The axes used for rotating are parallel to the x and y axes, but pass through the current segment origin.

Specifying $rx = 1$ and $ry = 0$ does not perform any rotation.

Because two zero values would be ambiguous, specifying $rx = 0$ and $ry = 0$ is taken as equivalent to $rx = 1$ and $ry = 0$ (no rotation).

dx,dy   Specify a displacement of dx parallel to the x axis, and dy parallel to the y axis. This transformation does not use the segment origin.

Specifying $dx = 0$ and $dy = 0$ does not perform any translation.

type   Specifies how the specified transformation should be used to modify the existing current model transformation (the existing transformation is the concatenation, in the current call context, of the instance, segment and model transformations, from the root segment downwards):-

0 - New/replace
         Any existing model transform is discarded and replaced by the combined effect of the specified components.

1 - Additive

> The combined effect of the specified components is added to the effect of the existing model transform, in the order (1) existing transform, (2) new transform.

2 - Preemptive

> The combined effect of the specified components is added to the effect of the existing model transform, in the order (1) new transform, (2) existing transform.

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_SEG_ID (NAME)
GPIERR_SEG_UNKNOWN
GPIERR_SEG_CALL_PRODUCES_RECURSIVE_LOOP
GPIERR_CALLED_SEG_NOT_FOUND
GPIERR_CALLED_SEG_IS_CURRENT
GPIERR_INVALID_TRANSFORM_TYPE
GPIERR_INVALID_TRANSFORM_PARAMETER
```

7.1.15.6 Viewing Transforms

GpiSetWindow

```
BOOL GpiSetWindow (hgpi, xl, xr, yb, yt)
HPS hgpi;
LONG xl;
LONG xr;
LONG yb;
LONG yt;
```

This sets the window, in Model Space, which corresponds to the viewport (see GpiSetViewport) in the Page. This defines the viewing transform.

This function is only valid outside segments. It applies, until changed, to all subsequently opened segments (it has no effect on primitives outside segments). All graphics primitives in a segment must have the same window. Once set for a segment, it can never be altered.

If the mapping of the window to the viewport (including the effect of any default viewing transform), and subsequently through the device transform, is such that one x-axis unit does not map to the same physical distance as one y-axis unit, the

picture will appear 'squashed', for example circles at the API will not appear to be circular.

Parameters:

hgpi	The handle of the Gpi presentation space.
xl,xr	The left and right boundaries of the window.
yb,yt	The bottom and top boundaries of the window.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_GRAPHICS_SEG_IS_CURRENT (I.E. OPEN)
GPIERR_INVALID_WINDOW_SPECIFICATION
```

GpiSetUniformWindow

```
BOOL GpiSetUniformWindow (hgpi, xl, xr, yb, yt)
HPS hgpi;
LONG xl;
LONG xr;
LONG yb;
LONG yt;
```

This sets the window, in Model Space, which corresponds to the viewport (see GpiSetViewport) in the Page. This defines the viewing transform.

The window is set such that either the x axis spans the entire width of the viewport and the y axis is within the height of the viewport, or that the y axis spans the entire height of the viewport and the x axis is within the width of the viewport. Thus, unless anisotropy has been deliberately built-in to the device transform (as, for example, with Page units of pels, on a device with non-square pels), one unit along x in Model Space will represent the same physical distance as one unit along y. If either axis is shorter than the width or height of the viewport, that axis is centred within the viewport.

GpiQueryWindow can subsequently be used to find the *actual* window bounds set as a result of this call.

This function is only valid outside segments. It applies, until changed, to all subsequently opened segments (it has no effect on primitives outside segments). All graphics primitives in a segment must have the same window. Once set for a segment, it can never be altered.

Parameters:

hgpi The handle of the Gpi presentation space.

xl,xr The left and right boundaries of the window.

yb,yt The bottom and top boundaries of the window.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_GRAPHICS_SEG_IS_CURRENT (I.E. OPEN)
GPIERR_INVALID_WINDOW_SPECIFICATION
```

GpiQueryWindow

```
BOOL GpiQueryWindow (hgpi, xl, xr, yb, yt)
HPS hgpi;
LONG *xl;
LONG *xr;
LONG *yb;
LONG *yt;
```

This returns the current window definition.

Parameters:

hgpi The handle of the Gpi presentation space.

*xl,*xr Variables which are set to the x co-ordinates of the left and right boundaries of the window.

*yb,*yt Variables which are set to the y co-ordinates of the bottom and top boundaries of the window.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
```

GpiSetViewport

```
BOOL GpiSetViewport (hgpi, xl, xr, yb, yt)
HPS hgpi;
LONG xl;
LONG xr;
LONG yb;
LONG yt;
```

This sets the viewport, in the Page, to which the window (see GpiSetWindow) will be mapped. This defines the viewing transform. GpiQueryPage can be issued to determine the extent of the Page.

A viewport is a subregion of the page. Viewports can be used to position the parts of a composite picture. The viewport boundaries are parallel to those of the page, space, and must be entirely within the page.

This function is only valid outside segments. It applies, until changed, to all subsequently opened segments (it has no effect on primitives outside segments). All graphics primitives in a segment must have the same viewport. Once set for a segment, it can never be altered.

Parameters:

hgpi	The handle of the Gpi presentation space.
xl,xr	The left and right boundaries of the viewport, in page units.
yb,yt	The bottom and top boundaries of the viewport, in page units.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_GRAPHICS_SEG_IS_CURRENT (I.E. OPEN)
GPIERR_UPPER_BOUNDARY_NOT_GREATER_THAN_LOWER
GPIERR_RIGHT_BOUNDARY_NOT_GREATER_THAN_LEFT
```

GpiQueryViewport

```
BOOL GpiQueryViewport (hgpi, xl, xr, yb, yt)
HPS hgpi;
LONG *xl;
LONG *xr;
LONG *yb;
LONG *yt;
```

This returns the current viewport definition.

Parameters:

hgpi	The handle of the Gpi presentation space.
*xl,*xr	Variables which are set to the x co-ordinates of the left and right boundaries of the viewport.
*yb,*yt	Variables which are set to the y co-ordinates of the bottom and top boundaries of the viewport.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION

GpiSetDefaultView

```
BOOL GpiSetDefaultView (hgpi, sx, sy, hx, hy,
        rx, ry, dx, dy, type)
HPS hgpi;
LONG sx;
LONG sy;
LONG hx;
LONG hy;
LONG rx;
LONG ry;
LONG dx;
LONG dy;
LONG type;
```

Sets the two-dimensional default viewing transform which is to apply to the whole picture.

The parameters specified are effectively combined into a single transformation matrix in the order scale, shear, rotation, displacement. This matrix is then used to update any previous default viewing transformation, depending upon the value of *type*.

If scaling values greater than unity are given, care must be taken that the combined effect of this and any other relevant transforms do not exceed fixed-point implementation limits. See the section, "Transform Range and Precision".

Parameters:

hgpi	The handle for the GPI presentation space.
sx,sy	A scale transformation in terms of an x-axis scaling (sx) and a y-axis scaling (sy). The origin of the scale is the origin of the page. If another scale origin is required, appropriate translations must be applied before and after. A scale factor of between 0 and 1 shrinks primitives; a scale factor greater than 1 stretches primitives. A negative scale factor reflects primitives about the other axis.
	The values are each represented as signed 32-bit values, with the low-order 16 bits taken to be to the right of the binary point. Specifying scale factors of 1 and 1 (actually, in this representation, 65536 and 65536) does not perform any scaling.
hx,hy	A shear transformation in terms of the displacements which a point on the y-axis makes after shearing. The axes used for shearing are parallel to the x and y axes, and pass through the page origin. Note that

primitives below the x axis are sheared in the opposite direction from those above the x axis. Points on the x axis itself are not moved. $hx = a$ and $hy = b$ produce an identical effect to $hx = -a$ and $hy = -b$.

Specifying $hx = 0$ and $hy = 1$ does not perform any shearing. Specifying $hy = 0$ is invalid, because it would produce an infinite shear.

rx,ry   A rotation transformation in terms of the displacements which a point on the x-axis makes after rotating. The axes used for rotating are parallel to the x and y axes, and pass through the page origin. If another rotation origin is required, appropriate translations must be applied before and after.

Specifying $rx = 1$ and $ry = 0$ does not perform any rotation.

Because two zero values would be ambiguous, specifying $rx = 0$ and $ry = 0$ is taken as equivalent to $rx = 1$ and $ry = 0$ (no rotation).

dx,dy   Specify a displacement of dx parallel to the x axis, and dy parallel to the y axis, in page units.

Specifying $dx = 0$ and $dy = 0$ does not perform any translation.

type   Specifies how the specified transformation should be used to modify the existing default viewing transformation, as follows:-

0 - New/replace
        The previous default viewing transform is discarded and replaced by the combined effect of the specified components.

1 - Additive
        The combined effect of the specified components is added to the effect of the existing default viewing transform, in the order (1) existing transform, (2) new transform. This option is most useful for incremental updates to transforms.

2 - Preemptive
        The combined effect of the specified components is added to the effect of the existing default viewing transform, in the order (1) new transform, (2) existing transform.

Returns:

    0 Error

1 OK

Principal errors:

```
GPIERR_INVALID_TRANSFORM_TYPE
GPIERR_INVALID_TRANSFORM_PARAMETER
```

7.1.15.7 Device Transform

GpiSetPageWindow

```
BOOL GpiSetPageWindow (hgpi, x, y, w, h)
HPS hgpi;
LONG x;
LONG y;
LONG w;
LONG h;
```

Sets the origin and size of the page window within the page.

The page window defines a window within the page. The corners of the page window are mapped to the corners of the page viewport, so the two together define the device transform.

It is not an error for any part of the page window to lie outside the page.

By default, the page window is coincident with the page.

Parameters:

hgpi   The handle for the GPI presentation space.

x,y   The origin of the page window in page units. If either is specified as *??*, the corresponding existing value is unchanged.

w,h   The width and height of the page window in page units. If either is specified as *??*, the corresponding existing value is unchanged. If either is specified as *0*, the corresponding page dimension is used.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_WINDOW_DEPTH_IS_INVALID
GPIERR_WINDOW_WIDTH_IS_INVALID
GPIERR_WINDOW_ROW_IS_INVALID
GPIERR_WINDOW_COLUMN_IS_INVALID
```

GpiQueryPageWindow

```
BOOL GpiQueryPageWindow (hgpi, x, y, w, h)
HPS hgpi;
LONG *x;
LONG *y;
LONG *w;
LONG *h;
```

This returns the origin and size of the page window.

Parameters:

hgpi	The handle for the GPI presentation space.
*x,*y	Variables in which the origin of the page window are returned.
*w,*h	Variables in which the width and height of the page window are returned.

Returns:

0 Error
1 OK

Principal errors:

-

GpiSetPageViewport

```
BOOL GpiSetPageViewport (hdc, x, y, w, h)
HPS hgpi;
LONG x;
LONG y;
LONG w;
LONG h;
```

Sets the origin and size of the page viewport within device space.

The page window (see GpiSetPageWindow) maps to the page viewport; together they define the device transform.

When a presentation space is associated with a Device Context, a default page viewport is set up, as described in the section, "Defaults and Examples".

Parameters:

hdc	The handle for the Device Context
x,y	The origin of the page viewport in device units. If either is specified as *??*, the corresponding existing value is unchanged.
w,h	The width and height of the page viewport in device units. If either is specified as *??*, the corresponding existing value is unchanged. If either is specified as *0*,

the corresponding default dimension is used.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_UPPER_BOUNDARY_NOT GREATER_THAN_LOWER
GPIERR_RIGHT_BOUNDARY_NOT_GREATER_THAN_LEFT

GpiQueryPageViewport

```
BOOL GpiQueryPageViewport (hdc, x, y, w, h)
HPS hgpi;
LONG *x;
LONG *y;
LONG *w;
LONG *h;
```

This returns the origin and size of the page viewport.

Parameters:

hdc	The handle for the Device Context
*x,*y	Variables in which the origin of the page viewport are returned.
*w,*h	Variables in which the width and height of the page viewport are returned.

Returns:

0 Error
1 OK

Principal errors:

-

7.1.15.8 Clipping

GpiBeginClipArea

```
BOOL GpiBeginClipArea (hgpi, control, mode)
HPS hgpi;
ULONG control;
LONG mode;
```

This introduces the definition of a clip area, which is terminated by a GpiEndClipArea function. The primitives between these cause no drawing to occur, but instead define a clip area. At GpiEndClipArea, this area is combined, in the manner specified by *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 (GpiBeginArea .. GpiEndArea) to occur within a clip area definition. If this occurs, the GpiBeginArea and GpiEndArea are effectively ignored, except that, within the area bracket, closure lines are generated as usual.

The co-ordinates are specified in world co-ordinates. A segment viewing limits (GpiSetViewingLimits) function causes a rectangular clip area to be set up.

At the start of each root segment, the clip area reverts to the default (infinite).

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

The following are the only Gpi functions allowed to the same presentation space, within a GpiBeginClipArea - GpiEndClipArea bracket:-

```
GpiBeginElement (containing only valid function(s))
GpiElement (containing a valid function)
GpiEndElement
GpiSetModelTransform
GpiCallSegment
GpiSetAttrMode
GpiQueryAttrMode
GpiPop (Providing only a valid function is popped)
GpiSetCurrentPosition
GpiQueryCurrentPosition
GpiMove
GpiLine
GpiPolyLine
GpiBox
GpiSetArcParams
GpiQueryArcParams
GpiArc
GpiFullArc
GpiPartialArc
GpiPolySpline
GpiPolyFillet
GpiPolyFilletSharp
GpiSetCharSet, GpiSetCharBox
GpiSetCharAngle, GpiSetCharShear
GpiSetCharDirection, GpiSetCharMode
GpiSetCharSpacing, GpiSetCharExtra, GpiSetCharBreakExtra
GpiSetTextAlignment
GpiCharString, GpiCharStringAt
GpiCharStringPos, GpiCharStringPosAt
```

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

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be

constructed and placed into the current segment.

Parameters:-

hgpi	The handle for the GPI presentation space
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.

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 specified 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

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_CLIP_AREA_CONTROL
GPIERR_INVALID_MODE
GPIERR_ATTEMPT_TO_START_SECOND_CLIP_AREA

GpiEndClipArea

```
BOOL GpiEndClipArea (hgpi)
HPS hgpi;
```

This terminates the definition of a clip area, introduced by a previous GpiBeginClipArea function.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:-

hgpi   The handle for the GPI presentation space

Returns:

0 Error
1 OK

Principal errors:

GPIERR_END_CLIP_AREA_DEFN_IGNORED

GpiSetGraphicsField

```
BOOL GpiSetGraphicsField (hgpi, x, y, w, h)
HPS hgpi;
LONG x;
LONG y;
LONG w;
LONG h;
```

This sets the mandatory clipping limits on the Page.

By default, the graphics field boundaries are coincident with the page boundaries. This function allows the mandatory clipping limits to be set to an area less than that of the whole page.

The boundaries are inclusive, ie a point on the boundary is not clipped.

Parameters:

hgpi   The handle for the GPI presentation space.

x,y   The origin of the graphics field in page space.

w,h   The width and height of the graphics field in page units.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_GRAPHICS_FIELD_ORIGIN
GPIERR_INVALID_GRAPHICS_WIDTH_OR_DEPTH

GpiQueryGraphicsField

```
BOOL GpiQueryGraphicsField (hgpi, x, y, w, h)
HPS hgpi;
LONG *x;
LONG *y;
LONG *w;
LONG *h;
```

This returns the dimensions of the graphics field.

Parameters:

hgpi   The handle for the GPI presentation space.

*x,*y   Variables in which the origin of the graphics field in page space are returned.

*w,*h   Variables in which the width and height of the graphics field in page units are returned.

Returns:

0 Error
1 OK

Principal errors:

-

GpiSetViewingLimits

```
BOOL GpiSetViewingLimits (hgpi, xl, xr, yb, yt)
HPS hgpi;
LONG xl;
LONG xr;
LONG yb;
LONG yt;
```

This sets the viewing limits of the current segment in Model Space.

The viewing limits, which are not subject to the current model or segment transform, define the boundaries to which subsequent primitives are to be clipped as they are drawn.

If this function is issued in a called segment, it overrides the values in force. However, the values are reset to defaults (no limit) at the start of the next root segment.

By specifying each of the four co-ordinates as zero, the viewing limits can be reset to their default values (no limit).

The boundaries are inclusive, ie a point on the boundary is not clipped.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Attribute Mode (see GpiAttribMode) has no effect on this function.

The default view limits are converted under the current Window-Viewport transform to a clipping rectangle in the page. This will remain in force until changed by a subsequent GpiSetViewingLimits function. Clipping actually takes place to the intersection of the viewing limits, the page window, the graphics field, and the client area on the device.

If not explicitly set, the default view limits cover all space.

Parameters:

hgpi   The handle for the GPI presentation space.

xl,xr   The x coordinates of the left and right limits.

yb,yt   The y coordinates of the bottom and top limits

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_SEG_VIEWING_LIMIT_SPECIFICATION
GPIERR_UPPER_BOUNDARY_NOT_GREATER_THAN_LOWER
GPIERR_RIGHT_BOUNDARY_NOT_GREATER_THAN_LEFT
```

GpiQueryViewingLimits

```
BOOL GpiQueryViewingLimits (hgpi, xl, xr, yb, yt)
HPS hgpi;
LONG *xl;
LONG *xr;
LONG *yb;
LONG *yt;
```

Returns the current viewing limits (see GpiSetViewingLimits).

This function is only valid for drawing modes of draw or draw-and-store.

Parameters:-

hgpi   The handle for the GPI presentation space.

*xl,*xr   Variables in which the x co-ordinates of the left and right limits are returned.

*yb,*yt   Variables in which the y co-ordinates of the bottom and top limits are returned.

Returns:

0 Error
1 OK

Principal errors:

-

7.1.15.9   Conversion Function

GpiConvert

```
BOOL GpiConvert (hgpi, src, targ, n, points)
HPS hgpi;
LONG src;
LONG targ;
LONG n;
GPOINT points[];
```

This converts an array of (x,y) co-ordinate pairs from one co-ordinate space to another.

The array contains x1, y1, x2, y2,... The co-ordinates are converted in situ.

Parameters:

hgpi   The handle for the GPI presentation space.

src,targ   Specify the source and target co-ordinate space, respectively, as follows:-

```
CVTC_WORLD         (1) World co-ordinates
CVTC_MODEL         (2) Model space
CVTC_DEFAULTPAGE   (3) Page space prior to default
                       viewing transform
CVTC_PAGE          (4) Page space after to default
                       viewing transform
CVTC_DEVICE        (5) Device space
```

n   The number of co-ordinate pairs in *points*.

points[]   An array of (x,y) co-ordinate pair structures.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_SRC_OR_TARG
GPIERR_INVALID_ARRAY_COUNT
```

GpiQueryViewportSize

```
BOOL GpiQueryViewportSize (hgpi, units, n, wh_array)
HPS hgpi;
ULONG units;
LONG n;
LONG wh_array[];
```

This calculates the sizes of viewports required (in page space), to correspond to specified window sizes, under the influence of the current transforms.

For example, if a window of 100 x 100 units of 0.1 mm is going to be used, and the current page units are 0.01 mm, this will return (1000,1000).

The array contains w1, h1, w2, h2,... - pairs of (width,height) window sizes. The co-ordinates are converted in situ to viewport sizes.

Parameters:

hgpi   The handle for the GPI presentation space.

units   This contains 32 bits (with bit 0 the least significant), in standard Intel format.

Bits (2-7) have the following meanings:-

PU_ISOTROPIC (B'000001')
    Arbitrary units, with the origin at the bottom left.

PU_PELS (B'000010')
    Pel co-ordinates, with the origin at the bottom left.

PU_LOWMETRIC (B'000011')
    Units of 0.1 mm, with the origin at the bottom left.

PU_HIMETRIC (B'000100')
    Units of 0.01 mm, with the origin at the bottom left.

PU_LOENGLISH (B'000101')
    Units of 0.01 in, with the origin at the bottom left.

PU_HIENGLISH (B'000110')
    Units of 0.001 in, with the origin at the bottom left.

PU_TWIPS (B'000111')
    Units of 1/1440 in, with the origin at the bottom left.

Other values are reserved. Other bits are also reserved and must be B'0'.

n   The number of co-ordinate pairs in *wh_array*.

wh_array[]   Specifies an array of (width, height) co-ordinate pairs.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_UNITS
GPIERR_INVALID_ARRAY_COUNT
```

7.1.16   General Attribute Functions

Functions described in this section are general functions for handling attributes.

7.1.16.1   Methods for Setting Attributes

Primitive attributes may be set in one of three ways:

1. Individually, as in GpiSetLineType

2. As a 'strip' of all attributes for a particular primitive type, as in GpiSetAttrs

3. Similar attributes may be set across all primitive types, as in GpiSetColor (this technique is only available for (foreground and background) color and mix attributes).

It makes no lasting difference which method is used; the second and third techniques may be looked upon as a kind of macro facility.

Individual attribute setting functions are described in the section of this document for the appropriate primitive type. 'Strip' functions are described in this section.

With color and mix attributes there is a third way of setting the attributes: across all primitive types. Thus GpiSetColor is a macro-like facility for setting the line color, the area color, the character color, etc. These functions are described in the section "Color and Mix Functions". There are no specific functions for setting individual color and mix attributes for specific primitive types - for example, there is no GpiSetLineColor function. The 'strip' technique can be used, however, to accomplish this (using an appropriate mask to indicate that only one attribute is being set).

If the third technique (eg GpiSetColor) is used in push-and-set mode (see below), each of the individual primitive values is pushed onto the stack. A single GpiPop function will suffice to recall all of them.

7.1.16.2   Default and Current Attributes

Each primitive attribute has a default value (not settable by the application), to which it is set at certain specific times, as described in the section, "Stored and Non-Stored Graphics Output". Also, the default version of any particular attribute may be requested, for example, by issuing GpiSetLineType with a parameter of *0*. This will cause the default value of that parameter to be copied to the current value.

7.1.16.3   Attribute Mode

Current values may either be *set,* or *push-and-set.* In the latter case the previous value is pushed onto a LIFO stack, before setting the attribute to the specified value. The old value may be reset either explicitly using Gpi-Pop, or implicitly by reaching the end of the segment in which it was pushed.

Whether a set or a push-and-set operation is performed is controlled by the attribute mode (see GpiSetAttrMode).

Micro-PS's do not support push-and-set mode. GpiSetAttrMode is invalid for a micro-PS.

7.1.16.4   Save and Restore Attributes

Functions are provided to cause various features of a presentation space, including the attributes, to be saved and restored. These functions operate with a stack which is independent of that used when the attribute mode is *push_ and_ set,* and also work for a micro-PS as well as a normal PS. See GpiSavePS and GpiRestorePS in the section, "Control Functions".

7.1.16.5   Attribute Queries

Functions are provided to return the current values of primitive attributes. If an attribute has been set to its default value, then the value returned will be the appropriate default value. For example, if the character direction attribute (for which the default is left-to-right) has never been changed, the GpiQueryCharDirection will return *0* (default) rather then *1* (left-to-right).

Primitive attribute querying functions are invalid if the drawing mode is store (and also in implcit draw mode).

7.1.16.6   Attribute Mode Functions

GpiSetAttrMode

```
BOOL GpiSetAttrMode (hgpi, mode)
HPS hgpi;
LONG mode;
```

Sets the API attribute mode which defines whether, when attributes are set, the old value should first be pushed to the segment call stack. If this is done, the value may be popped from the stack either explicitly using GpiPop, or implicitly at the end of the segment. The latter technique may be used, for example, to ensure that when a called segment terminates, the primitive attributes are the same as they were when it was called.

This is an API mode. Which mode to use for a particular GpiSet... function is decided by the attribute mode current at the time the GpiSet... function is passed across the API. The mode may be changed at any time, and will not affect any attribute setting functions which have already been stored in the segment store.

Attribute mode only applies to attributes passed across the API in individual GpiSet... functions (including GpiSetAttrs and functions such as GpiSetColor). It does not apply to any attribute setting functions passed across in bulk, such as MetPlayMetaFile; these already indicate individually whether they should cause the old value to be pushed.

Parameters:

hgpi   The handle for the GPI presentation space.

       This must be a normal PS, not a micro-PS.

mode   The attribute mode, as follows:-

```
0 Push attributes before setting (default)
1 Do not push attributes before setting
```

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_INVALID_ATTR_MODE
```

GpiQueryAttrMode

```
LONG GpiQueryAttrMode (hgpi)
HPS hgpi;
```

This returns the current setting of the attribute mode (see GpiSetAttrMode).

Parameters:-

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Current attribute mode (see GpiSetAttrMode.)

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION

GpiPop

```
BOOL GpiPop (hgpi, count)
HPS hgpi;
LONG count;
```

Pops, from the segment call stack, the most recent attribute or control that was pushed by the current segment (see GpiSetAttrMode).

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi   The handle for the GPI presentation space.

          This must be a normal PS, not a micro-PS.

count   The number of attributes which are to be popped from the stack. (Note that GpiSetAttrs pushes as many attributes onto the stack as were indicated at the time.)

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MICROPS_FUNCTION
GPIERR_PRIMITIVE_STACK_EMPTY
GPIERR_INVALID_ATTR_COUNT

same as 1
but any that are

7.1.16.7   Attribute Strip Setting Functions

GpiSetAttrs

```
BOOL GpiSetAttrs (hgpi, prim_type, attrs_mask,
        defs_mask, attrs)
HPS hgpi;
LONG prim_type;
ULONG attrs_mask;
ULONG defs_mask;
LPBUF attrs;
```

Sets attributes for the specified primitive type.

Any attribute (for the specified primitive type) for which the appropriate flag is set in the *attrs_ mask* has its value updated:-

- If the corresponding flag in *defs_ mask* is also set, the attribute is set to default.

- If the corresponding flag in *defs_ mask* is not set, then the attribute is set to the value specified in the *attrs* structure.

Any attribute for which the appropriate flag in *attrs_ mask* is not set is unchanged, regardless of the setting of the corresponding flag in *defs_ mask*.

The *defs_ mask* and *attrs_ mask* parameters each consist of 32 flags. Each attribute of the primitive type in question is represented by one flag. The flag numbering, which is the same in both fields, is described below.

The data in the *attrs* buffer consists of a structure of attribute data. The layout of the structure is fixed (see below) for each primitive type. Only data for attributes for which the flag is set in *attrs_ mask* (but not in *defs_ mask*) will be inspected; any other data will be ignored (indeed, the buffer need be no longer than is necessary to contain the data for the highest offset attribute referenced).

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the functions (one for each attribute, in the order in which the attributes are specified) are generated.

Parameters:

hgpi   The handle for the GPI presentation space.

prim_ type
            The primitive type for which attributes are to be set, as follows:-

```
BATTR_LINE     1 - Line and arc primitives
BATTR_CHAR     2 - Character primitives
BATTR_MARKER   3 - Marker primitives
BATTR_PATTERN  4 - Area primitives (pattern)
BATTR_IMAGE    5 - Image primitives
```

attrs_ mask

A 4-byte parameter containing 32 flags (see below for numbering). Each flag which is set indicates that either the corresponding flag in *defs_ mask* is set, or that the *attrs* buffer contains data for the corresponding attribute. If all of the flags in *attrs_ mask* are zero, the *attrs* buffer address is not used.

defs_ mask

A 4-byte parameter containing 32 flags (see below for numbering). Each flag which is set (and for which *attrs_ mask* is also set) causes the corresponding attribute to be set to its default value.

attrs

A buffer containing attribute values for each attribute for which the *attrs_ mask* flag was set, at the correct offset as specified below for the particular primitive type.

- *Line attributes* :-

  attrs: -

  ```
  struct LINEBUNDLE {
       LONG    color;
       LONG    back_color;
       UINT    mix_mode;
       UINT    back_mix_mode;
       ULONG   width;
       LONG    geom_width;
       UINT    type;
       UINT    end;
       UINT    join;
       UINT    set;
       UINT    symbol;
  };
  ```

  attrs_ mask and *defs_ mask* bits: -

  ```
  LBB_COLOR               0x00000001L
  LBB_BACK_COLOR          0x00000002L
  LBB_MIX_MODE            0x00000004L
  LBB_BACK_MIX_MODE       0x00000008L
  LBB_TYPE                0x00000010L
  LBB_WIDTH               0x00000020L
  LBB_GEOM_WIDTH          0x00000040L
  LBB_END                 0x00000080L
  LBB_JOIN                0x00000100L
  LBB_PATTERN_SET         0x00000200L
  LBB_PATTERN_SYMBOL      0x00000400L
  ```

- *Character attributes* :-

  attrs : -

  ```
  struct CHARBUNDLE{
        LONG     color;
        LONG     back_color;
        UINT     mix_mode;
        UINT     back_mix_mode;
        UINT     set;
        UINT     precision;
        LONG     cell[2];
        LONG     angle_xy[2];
        LONG     shear_xy[2];
        LONG     text_align[2];
        LONG     spacing[2];
        UINT     direction;
        UINT     break_extra1;
        LONG     break_extra2;
        LONG     break_extra3;
        LONG     extra[2];
  } CHARBUNDLE;          /* cbnd */
  ```

  attrs_ mask and *defs_ mask* bits : -

CBB_COLOR	0x00000001L
CBB_BACK_COLOR	0x00000002L
CBB_MIX_MODE	0x00000004L
CBB_BACK_MIX_MODE	0x00000008L
CBB_SET	0x00000010L
CBB_MODE	0x00000020L
CBB_BOX	0x00000040L
CBB_ANGLE_XY	0x00000080L
CBB_SHEAR_XY	0x00000100L
CBB_ALIGNMENT	0x00000200L
CBB_SPACING	0x00000400L
CBB_DIRECTION	0x00000800L
CBB_BREAK_EXTRA	0x00001000L
CBB_EXTRA	0x00002000L

Text alignment contains two 2-byte values.

Character break extra is a 2-byte code point, followed by 4-byte horizontal and vertical values.

- *Marker attributes* :-

  attrs : -

  ```
  struct MARKERBUNDLE{
        LONG     color;
        LONG     back_color;
        UINT     mix_mode;
        UINT     back_mix_mode;
        UINT     set;
        UINT     symbol;
        LONG     cell[2];
  };
  ```

attrs_ mask and *defs_ mask* bits:-

```
MBB_COLOR                0x00000001L
MBB_BACK_COLOR           0x00000002L
MBB_MIX_MODE             0x00000004L
MBB_BACK_MIX_MODE        0x00000008L
MBB_SET                  0x00000010L
MBB_SYMBOL               0x00000020L
MBB_BOX                  0x00000040L
```

- *Pattern attributes* (note that pattern reference point is not settable in this way):-

  attrs:-

```
struct PATTERNBUNDLE{
    LONG    color;
    LONG    back_color;
    UINT    mix_mode;
    UINT    back_mix_mode;
    UINT    set;
    UINT    symbol;
};
```

  attrs_ mask and *defs_ mask* bits:-

```
PBB_COLOR                0x00000001L
PBB_BACK_COLOR           0x00000002L
PBB_MIX_MODE             0x00000004L
PBB_BACK_MIX_MODE        0x00000008L
PBB_SET                  0x00000010L
PBB_SYMBOL               0x00000020L
```

- *Image attributes* :-

  attrs:-

```
struct IMAGEBUNDLE{
    LONG    color;
    LONG    back_color;
    UINT    mix_mode;
    UINT    back_mix_mode;
};
```

  attrs_ mask and *defs_ mask* bits:-

```
IBB_COLOR                0x00000001L
IBB_BACK_COLOR           0x00000002L
IBB_MIX_MODE             0x00000004L
IBB_BACK_MIX_MODE        0x00000008L
```

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_FOREGROUND_BACKGROUND_MIX_COMBINATION
GPIERR_INVALID_PRIMITIVE_TYPE

```
GPIERR_ATTRS_MASK_SPECIFIES_UNSUPPORTED_ATTRS
GPIERR_DEFS_MASK_SPECIFIES_UNSUPPORTED_ATTRS
GPIERR_INVALID_FUNCTION_IN_VECTOR_SYMBOL_SET
GPIERR_INVALID_FUNCTION_IN_IMAGE_DEFN
GPIERR_INVALID_FUNCTION_IN_ROOT_SEG_PROLOG
GPIERR_INVALID_COLOR_ATTR
GPIERR_INVALID_BACKGND_COLOR_ATTR
GPIERR_INVALID_MIX_ATTR
GPIERR_INVALID_BACKGROUND_MIX_ATTR
GPIERR_INVALID_LINE_TYPE_ATTR
GPIERR_INVALID_LINE_WIDTH_ATTR
GPIERR_INVALID_GEOM_LINE_WIDTH_ATTR
GPIERR_INVALID_LINE_END_ATTR
GPIERR_INVALID_LINE_JOIN_ATTR
GPIERR_INVALID_LINE_PATTERN_SET_ATTR
GPIERR_INVALID_LINE_PATTERN_ATTR
GPIERR_INVALID_PATTERN_SET_ATTR
GPIERR_INVALID_PATTERN_ATTR
GPIERR_INVALID_CHAR_SET_ATTR
GPIERR_INVALID_CHAR_BOX_ATTR
GPIERR_INVALID_CHAR_ANGLE_ATTR
GPIERR_INVALID_CHAR_SHEAR_ATTR
GPIERR_INVALID_CHAR_DIRECTION_ATTR
GPIERR_INVALID_CHAR_MODE_ATTR
GPIERR_INVALID_CHAR_SPACING_ATTR
GPIERR_INVALID_CHAR_EXTRA_ATTR
GPIERR_INVALID_CHAR_BREAK_EXTRA_ATTR
GPIERR_INVALID_TEXT_ALIGN_ATTR
GPIERR_INVALID_MARKER_SET_ATTR
GPIERR_INVALID_MARKER_ATTR
GPIERR_INVALID_MARKER_BOX_ATTR
```

GpiQueryAttrs

```
BOOL GpiQueryAttrs (hgpi, prim_type, attrs_mask,
        defs_mask, attrs)
HPS hgpi;
LONG prim_type;
ULONG attrs_mask;
ULONG *defs_mask;
LPBUF attrs;
```

Returns current attributes for the specified primitive type.

This function is only valid for draw or draw-and-store modes.

If the value of a requested attribute is 'default', then as well as returning data in *attrs*, the corresponding bit in *defs_ mask* is turned on. Otherwise, the flag in *defs_ mask* is reset.

The parameters returned by GpiQueryAttrs may be used to reinstate exactly the same attributes as were queried, using GpiSetAttrs.

Parameters:

hgpi The handle for the GPI presentation space.

prim_type
   The primitive type for which attributes are to be set,
   as follows:-

```
BATTR_LINE     1 - Line and arc primitives
BATTR_CHAR     2 - Character primitives
BATTR_MARKER   3 - Marker primitives
BATTR_PATTERN  4 - Area primitives (pattern)
BATTR_IMAGE    5 - Image primitives
```

attrs_mask
   A 4-byte parameter containing 32 flags (see
   GpiSetAttrs for numbering). Each flag which is set
   indicates that the corresponding flag in *defs_mask*
   should be updated, and that the value of the attribute
   should be returned in the *attrs* buffer.

   If all of the flags in *attrs_mask* are zero, the *attrs*
   buffer address is not used.

*defs_mask
   A 4-byte parameter containing 32 flags (see
   GpiSetAttrs for masks).

   Each flag in *defs_mask* is updated if the corresponding
   flag in *attrs_mask* was set. It is set if the attribute is
   set to the default, and reset otherwise. Other flags are
   undefined.

attrs A buffer in which is returned the value of each attri-
   bute for which the *attrs_mask* flag was set, in the
   order specified in GpiSetAttrs for the particular primi-
   tive type.

   Only data for attributes for which the appropriate flag
   in *attrs_mask* is set is updated, so *attrs* need only be as
   long as required for the highest offset attribute to be
   returned (see GpiSetAttrs).

Returns:

 0 Error
 1 OK

Principal errors:

```
GPIERR_INVALID_PRIMITIVE_TYPE
GPIERR_ATTRS_MASK_SPECIFIES_UNSUPPORTED_ATTRS
```

7.1.17 Color and Mix Functions

This section describes functions available for loading a logical color table, and for manipulating the foreground and background color and mix attributes.

Note that there are separate color and mix attributes for each primitive type. The color and mix setting functions described here set the appropriate values across all primitive types. They are provided for applications which do not require to distinguish the values between primitive types, and prefer to view them as global attributes. The query functions actually return the values for line primitives.

The individual primitive types are as follows:-

● Line and arc primitives

● Character primitives

● Marker primitives

● Area primitives

● Image primitives

Default values of these attributes are (for all primitive types):-

```
Color:                Color 7
Background color:     Color 0
Mix:                  Overpaint
Background mix:       Leave alone
```

For further information on the treatment of color in GPI, see the section, "Color".

7.1.17.1 Resources and Default Functions

GpiCreateLogColorTable

```
BOOL GpiCreateLogColorTable (hgpi, options, format,
     start, count, data)
HPS hgpi;
ULONG options;
LONG format;
LONG start;
LONG count;
LPBUF data;
```

This function defines the entries of the logical color table.

It may cause the color table to be preset to the default values. These are:-

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)

Color 0 is special in that it is the one which GpiErase clears to. Colors beyond 7 have device-dependent defaults.

Parameters:

hgpi	The handle for the GPI presentation space.
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 GpiRealizeColorTable 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, GpiRealizeColorTable may have no effect
>
> Other flags are reserved and must be B'0'.

format	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

start	starting index (only relevant for LCOLF_CONSECRGB)
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.

data The start address of 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.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_COLOR_OPTIONS
GPIERR_INVALID_COLOR_FORMAT
GPIERR_INVALID_START_INDEX
GPIERR_INVALID_ARRAY_COUNT
GPIERR_INVALID_COLOR_DATA
```

GpiRealizeColorTable

```
BOOL GpiRealizeColorTable (hgpi)
HPS hgpi;
```

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.

If the presentation space is currently associated with a screen window device, then this function should only be issued when the associated window is maximized.

Parameters:

hgpi The handle for the GPI presentation space.

Returns:

0 Error
1 OK

Principal errors:

-

GpiUnrealizeColorTable

```
BOOL GpiUnrealizeColorTable (hgpi)
HPS hgpi;
```

This function is the reverse of GpiRealizeColorTable. It should be issued when the associated window ceases to be maximised. It causes the default color table to be reinstated.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

```
0 Error
1 OK
```

Principal errors:

-

GpiQueryColorData

```
BOOL GpiQueryColorData (hgpi, count, array)
HPS hgpi;
LONG count;
LONG array[];
```

Returns information about the currently available color table and device colors.

Parameters:

hgpi   The handle for the GPI presentation space.

count   The number of elements supplied in *array*.

array[count]
          An array which on return will contain:-

array[0] Format of loaded color table if any:-

          LCOLF_DEFAULT (0)
                    Default color table is in force.

          LCOLF_INDRGB (1)
                    Color table loaded which provides translation from index to RGB.

          LCOLF_RGB (3)
                    Color index = RGB.

array[1] Smallest color index loaded (*0* if the default color table is in force)

array[2] Largest color index loaded (*0* if the default color table is in force)

array[3] Maximum number of distinct colors available at one time

array[4]   Maximum number of distinct colors
specifiable on device

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

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_ARRAY_COUNT
```

GpiQueryLogColorTable

```
LONG GpiQueryLogColorTable (hgpi, options, start,
     count, array)
HPS hgpi;
ULONG options;
LONG start;
LONG count;
LONG array[];
```

Returns the logical color table currently associated device, one
at a time.

Parameters:

hgpi   The handle for the GPI presentation space.

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'.

start   The starting index for which data is to be returned.

count   The number of elements available in *array*.

array[count]
        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 GpiCreateLogColorTable),
        starting with the specified index, and ending either
        when there are no further loaded entries in the table,
        or when *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 = B'1', it is an array of alternating

color indices and values, in the order index1, value1, index2, value2,... An even number of elements will always be returned in this case. If the logical color table is not loaded with a contiguous set of indices, any index values which are not loaded will be skipped.

Returns:

-1 Error
>=0 Number of elements returned

Principal errors:

GPIERR_INVALID_COLOR_OPTIONS
GPIERR_INVALID_START_INDEX
GPIERR_INVALID_ARRAY_COUNT

GpiQueryRealColors

```
LONG GpiQueryRealColors (hgpi, options, start, count, array)
HPS hgpi;
ULONG options;
LONG start;
LONG count;
LONG array[];
```

Returns the rgb values of the distinct colors available on the currently associated device, one at a time.

Parameters:

hgpi	The handle for the GPI presentation space.
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'.

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.
count	The number of elements available in *array.*

array[count]
>   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 Gpi-CreateLogColorTable).

>   If LOPT_ INDEX = B'1', it is an array of alternating color indices and values, in the order index1, value1, index2, value2,... An even number of elements will always be returned in this case.

Returns:

-1 Error
>=0 Number of elements returned

Principal errors:

```
GPIERR_INVALID_COLOR_OPTIONS
GPIERR_START_INDEX
GPIERR_INVALID_ARRAY_COUNT
```

GpiQueryNearestColor

```
LONG GpiQueryNearestColor (hgpi, options, in_rgb_color)
HPS hgpi;
ULONG options;
LONG in_rgb_color;
```

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

Parameters:

hgpi   The handle for the GPI presentation space.

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'.

in_ rgb_ color
>   The required color

Returns:

-1 Error
>=0 Nearest available color to the one specified

Principal errors:

```
GPIERR_INVALID_COLOR_OPTIONS
GPIERR_INVALID_RGBCOLOR
```

GpiQueryColorIndex

```
LONG GpiQueryColorIndex (hgpi, options, rgb_color)
HPS hgpi;
ULONG options;
LONG rgb_color;
```

This returns the color index of the device color which is closest to the specified RGB color representation, for the device connected to the specified presentation space.

Parameters:

hgpi   The handle for the GPI presentation space.

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'.

rgb_ color
Specifies a color in RGB terms

Returns:

```
-1 Error
>=0 Color index providing closest match to the
        specified color
```

Principal errors:

```
GPIERR_INVALID_COLOR_OPTIONS
GPIERR_INVALID_RGBCOLOR
```

GpiQueryRGBColor

```
LONG GpiQueryRGBColor (hgpi, options, color)
HPS hgpi;
ULONG options;
LONG color;
```

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

Parameters:

hgpi   The handle for the GPI presentation space.

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'.

color   Specifies a color index

Returns:

```
-1 Error
>=0 RGB color providing closest match to the
       specified color index
```

Principal errors:

```
GPIERR_INVALID_COLOR_OPTIONS
GPIERR_INVALID_COLOR_INDEX
```

7.1.17.2   Attribute Setting Functions

GpiSetColor

```
BOOL GpiSetColor (hgpi, color)
HPS hgpi;
LONG color;
```

Sets the current color index attribute, for each individual primitive type, to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The current values for each primitive type will be updated. The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated (in this case the values of the color attribute for each primitive type are pushed. A single GpiPop will restore them).

Parameters:

hgpi   The handle for the GPI presentation space

color   The color index value required, as follows:-

```
-3 - set to default value
-2 - equivalent to 7 (display), or 0 (printer)
       (ie white for default color table)
-1 - equivalent to 0 (display), or 7 (printer)
```

```
                             (ie black for default color table)
                 0-n - color index value
```

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_COLOR_ATTR
```

GpiQueryColor

```
LONG GpiQueryColor (hgpi)
HPS hgpi;
```

Returns the current (line) color index attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

```
-4 Error
>=-3 Color index
```

Principal errors:

```
-
```

GpiSetBackColor

```
BOOL GpiSetBackColor (hgpi, color)
HPS hgpi;
LONG color;
```

Sets the current background color index attribute, for each individual primitive type, to the specified value.

Note that if the background mix is transparent, the default setting, the background color will not be seen.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated (in this case the values of the background color attribute for each primitive type are pushed. A single GpiPop will restore them).

Parameters:

hgpi   The handle for the GPI presentation space.

color   The color index value required, as follows:-

```
-3 - set to default value
-2 - equivalent to 7 (display), or 0 (printer)
     (ie white for default color table)
-1 - equivalent to 0 (display), or 7 (printer)
     (ie black for default color table)
0-n - color index value
```

Returns:

```
0 Error
1 OK
```

GpiQueryBackColor

```
LONG GpiQueryBackColor (hgpi)
HPS hgpi;
```

Returns the current (line) background color index attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*color   A variable in which the (line) background color index attribute is returned.

Returns:

```
-4 Error
>=-3 Background color index
```

Principal errors:

-

GpiSetMix

```
BOOL GpiSetMix (hgpi, mix_mode)
HPS hgpi;
LONG mix_mode;
```

Sets the current mix attribute, for each individual primitive type, to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated (in this case the values of the mix attribute for each primitive type are pushed. A single GpiPop will restore them).

Parameters:

hgpi   The handle for the GPI presentation space.

mix_ mode

The required mix mode. Valid values for *mix_ mode* are:

```
MIX_DEFAULT       ( 0) - Use default
MIX_OR            ( 1) - Or (*)
MIX_OVERPAINT     ( 2) - Overpaint (*)
MIX_UNDERPAINT    ( 3) - Underpaint
MIX_XOR           ( 4) - XOR
MIX_LEAVEALONE    ( 5) - Leave alone (invisible) (*)
MIX_AND           ( 6) - AND
MIX_SUBTRACT      ( 7) - (inverse source) AND dest
MIX_MASKSRCNOT    ( 8) - Source AND (inverse dest)
MIX_ZERO          ( 9) - All zeroes
MIX_NOTMERGESRC   (10) - Inverse (source OR dest)
MIX_NOTXORSRC     (11) - Inverse (source XOR dest)
MIX_INVERT        (12) - Inverse (dest)
MIX_MERGESRCNOT   (13) - Source OR (inverse dest)
MIX_NOTCOPYSRC    (14) - Inverse (source)
MIX_MERGENOTSRC   (15) - Inverse (source) OR dest
MIX_NOTMASKSRC    (16) - Inverse (source AND dest)
MIX_ONE           (17) - All ones
```

Mixes marked with an asterisk (*) are mandatory for all devices, except that OR is only mandatory for devices capable of supporting it. Unsupported mix values will produce overpaint. DevQueryCaps can be used to determine the level of support.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MIX_ATTR

GpiQueryMix

```
LONG GpiQueryMix (hgpi)
HPS hgpi;
```

Returns the current (line) mix attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Mix mode

Principal errors:

GpiSetBackMix

```
BOOL GpiSetBackMix (hgpi, mix_mode)
HPS hgpi;
LONG mix_mode;
```

Sets the current background mix attribute, for each individual primitive type, to the specified value.

Note that if the background mix is transparent, the default setting, the background mix will not be seen.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated (in this case the values of the background mix attribute for each primitive type are pushed. A single GpiPop will restore them).

The following primitives are affected by the background mix attribute:-

Areas   The background of an area is defined to be every pixel within the area that is not set by the shading pattern.

Mode-1 and Mode-2 Text
        The background of a mode-1 or mode-2 character is every pixel within the character definition that is not set.

Mode-3 Text
        The background of a mode-3 character is the complete character box.

Images   For an image, the background is every pixel within the image that is not set.

Markers
        For an image marker, the background is every pixel within the marker definition that is not set. The background of a vector marker is the complete marker box.

Parameters:

hgpi   The handle for the GPI presentation space.

mix_ mode
        Valid values for *mix_ mode* are:

```
MIX_DEFAULT      ( 0) - Use default
MIX_OR           ( 1) - Or (*)
MIX_OVERPAINT    ( 2) - Overpaint (*)
MIX_UNDERPAINT   ( 3) - Underpaint
```

```
MIX_XOR            ( 4) - XOR
MIX_LEAVEALONE     ( 5) - Leave alone (invisible) (*)
```

Background mixes marked with an asterisk (*) are mandatory for all devices. Unsupported mix values will produce transparent. DevQueryCaps can be used to determine the level of support.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_BACKGROUND_MIX_ATTR
GPIERR_INVALID_FOREGROUND_BACKGROUND_MIX_COMBINATION
```

GpiQueryBackMix

```
LONG GpiQueryBackMix (hgpi)
HPS hgpi;
```

Returns the current (line) background mix attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Background mix mode

Principal errors:

-

7.1.18 Line Functions

Functions described in this section are for drawing lines and boxes, and setting up line attributes. The attributes described also affect arc functions described in the section, "Arc Functions".

Lines, boxes and arcs are drawn with the following attributes:-

- Line color
- Line background color
- Line mix
- Line background mix

- Line type
- Line width
- Line geometric type
- Line geometric width
- Line end
- Line join
- Line pattern set
- Line pattern symbol

These attributes may either be set individually, or in a single function.

The manipulation of the color, background color, mix, and background mix attributes globally is described in the section, "Color and Mix Functions".

The default values of line attributes are as follows:-

```
Line color:              Color 7
Line background color:   Color 0
Line mix:                Overpaint
Line background mix:     Leave alone
Line type:               Solid
Line width:              Normal
Line width geometric:    Use cosmetic width
Line type geometric:     Solid
Line end:                Flat
Line join:               Bevel
Line pattern set:        Standard 17-value set
Line pattern symbol:     Solid shading
```

7.1.18.1 Resources and Defaults Functions

GpiLoadLineType

```
BOOL GpiLoadLineType (hgpi, length, data)
HPS hgpi;
LONG length;
LPBUF data;
```

Loads one or more line-type definitions, from a specified application data area, into the presentation space.

These are *cosmetic* line types, which are used when the current geometric line width is not either *-1* or *0*.

When a new definition is provided for an already loaded codepoint, the existing definition will be replaced.

Parameters:

hgpi	The handle for the GPI presentation space.
length	The length of the *data* structure.
data	The address of the data area containing the line-type definition(s). The format is as follows:-

Byte 0   Flags1

> *Bit 7* '1'B specifies clear all definitions,
> and set table_id = 'FFFF'X, '0'B speci
> do not clear (add to / replace definit
> (i.e. '80'X specifies clear).

Byte 1   Flags2

Reserved - must be zero.

Bytes 2 & 3
table_ id

- 'FFFF'X specifies leave the table_ id unchanged

- Values in the range '0000'X-'00FE'X specify change it to this value.

- Other values are invalid.

This is followed by self defining parameters with the following format:

Byte 0   (n + 1) Length of Self Defining Parameter

Byte 1   Type '01'X = Media Pel Image

Byte 2   Line type codepoint of definition, in range 65 - 254 inclusive

Byte 3   Reserved '00'X

Bytes 4-n
Counts in pairs of bytes; first byte = pels on, second byte = pels off.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_OR_INCONSISTENT_LENGTHS
GPIERR_INVALID_LINE_TYPE_CODEPOINT
GPIERR_INVALID_RESERVED_FIELD
GPIERR_INVALID_LTTID
GPIERR_INCOMPLETE_LINE_DEFN
```

GpiQueryLineTypes

```
LONG GpiQueryLineTypes (hgpi, length, data)
HPS hgpi;
LONG length;
LPBUF data;
```

This returns a record providing details of all available cosmetic
line types, including both standard ones and loaded ones.

Parameters:

hgpi	The handle for the GPI presentation space.
length	The length of the *data* buffer
data	A pointer to a data buffer in which the line type data is to be returned. The data is returned as a record consisting of the length of data returned in bytes followed by the data.

The format is as follows:-

Byte 0	'00'X Flags - reserved
Byte 1	'09'X Number of Pairs
Byte 2	'00'X attribute = zero (default)
Byte 3	'07'X action = solid
Byte 4	'01'X attribute = 1
Byte 5	'01'X action = dotted
Byte 6	'02'X attribute = 2
Byte 7	'02'X action = short dashed
Byte 8	'03'X attribute = 3
Byte 9	'03'X action = dash, dot
Byte 10	'04'X attribute = 4
Byte 11	'04'X action = double dotted
Byte 12	'05'X attribute = 5
Byte 13	'05'X action = long dashed
Byte 14	'06'X attribute = 6
Byte 15	'06'X action = dash, double dot
Byte 16	'07'X attribute = 7
Byte 17	'07'X action = solid

Byte 18   '08'X attribute = 8

Byte 19   '08'X action = invisible

Byte 20   '04'X Length of (Loadable Formats) self defining parameter

Byte 21   '01'X Self defining parameter type (Loadable Formats)

Byte 22   '40'X Format Type (media pel image)

Byte 23   '00'X Reserved

Byte 24   (n - 23) Length of (Loaded Line Types) self defining parameter including this byte

Byte 25   '02'X Self defining parameter type (Loaded Line Types)

Bytes 26 & 27
        table_ id ('FFFF'X if not loaded).

Bytes 28-n
        List of already loaded codepoints (one byte per codepoint)

Returns:

-1 Error
>=0 Length of data returned

Principal errors:

GPIERR_INVALID_LENGTH

7.1.18.2   Attribute Setting Functions

GpiSetLineType

```
BOOL GpiSetLineType (hgpi, line_type)
HPS hgpi;
LONG line_type;
```

Sets the current cosmetic line-type attribute to the specified value. A non-solid line-type consists of a sequence of 'on' and 'off' runs which gives the appearance of a dotted, dashed, etc line.

This attribute specifies the *cosmetic* line type, which is used when the current geometric line width is not either *-1* or *0*.

The 8 standard line-types are implemented on each device to give a good appearance on that device, taking into account the pel resolution. Application-loaded line-type definitions are defined in terms of pel runs, and may need to be changed

between devices with very different pel resolutions.

The system maintains position within the line-type definition (either standard or application-loaded), 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:-

- GpiSetLineType
- GpiMove
- GpiSetCurrentPosition
- GpiCallSegment
- GpiSetSegmentTransform
- GpiSetModelTransform
- GpiSetWindow
- GpiSetUniformWindow
- GpiSetViewport
- GpiSetPageWindow
- GpiSetPageViewport
- GpiPop (or end of called segment) which pops current position or a model transform

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment. will be updated.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

line_ type
         Valid values for *line_ type* are:-

```
LINETYPE_DEFAULT        (0) - Use default
LINETYPE_DOT            (1) - Dotted
LINETYPE_SHORTDASH      (2) - Short dashed
LINETYPE_DASHDOT        (3) - Dash dot
LINETYPE_DOUBLEDOT      (4) - Double dot
LINETYPE_LONGDASH       (5) - Long dash
LINETYPE_DASHDOUBLEDOT  (6) - Dash, double dot
LINETYPE_SOLID          (7) - Solid
LINETYPE_INVISIBLE      (8) - Invisible
                        (65 through 254) - User
                        defined line types
```

If the specified line type is not valid, the default type

is used.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_LINE_TYPE_ATTR

GpiQueryLineType

```
LONG GpiQueryLineType (hgpi)
HPS hgpi;
```

Returns the current cosmetic line-type attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Line type

Principal errors:

-

GpiSetLineWidth

```
GpiSetLineWidth (hgpi, line_width)
HPS hgpi;
LONG line_width;
```

Sets the current cosmetic line-width attribute to the specified value.

The cosmetic line width specifies a multiplier on the 'normal' line thickness for the device. Cosmetic thickness does not depend upon transforms, so that, for example, lines do not become thicker if a 'zoom in' occurs.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

line_ width

> The required line width. 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 width of 1.0.

```
LINEWIDTH_DEFAULT    (0) - Use default
LINEWIDTH_NORMAL     (1) - Normal width
LINEWIDTH_THICK      (2) - Thick (double width)
```

> Any other positive value is a multiplier on the 'normal' line width. In many implementations values $<= 1.0$ (other than 0.0) will be treated as 'normal', and values > 1.0 as 'thick'.

Returns:

> *0* Error
> *1* OK

Principal errors:

GPIERR_INVALID_LINE_WIDTH_ATTR

GpiQueryLineWidth

```
LONG GpiQueryLineWidth (hgpi)
HPS hgpi;
LONG x;
```

Returns the current line-width attribute.

The value returned is a 4-byte fixed-point number with the high-order word as the integer portion and the low-order word as the fractional portion (see GpiSetLineWidth).

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

> *-1* Error
> $>=0$ Line width

Principal errors:

> –

GpiSetLineWidthGeom

```
BOOL GpiSetLineWidthGeom (hgpi, line_width)
HPS hgpi;
LONG line_width;
```

Sets the current geometric line-width attribute to the specified

value.

The geometric line width is specified in Drawing Order Co-ordinate Space units, so that, for example, the thickness varies on a zoom operation.

Setting the geometric line thickness (to > 0) also brings the geometric line type into effect (see GpiSetLineTypeGeom).

If more than one API function is needed to define a thick line construct, the functions should be enclosed within a strokes bracket. See GpiBeginStrokes.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be con-structed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

line_ width
           A signed long integer containing the required geometric line width. Valid values are:-

> *0* Use default
> *-1* Use cosmetic line width
> *>0* Thickness in world co-ordinates

Returns:

> *0* Error
> *1* OK

Principal errors:

GPIERR_INVALID_GEOM_LINE_WIDTH_ATTR
GPIERR_STROKES_CONTEXT_ERROR

GpiQueryLineWidthGeom

```
LONG GpiQueryLineWidthGeom (hgpi)
HPS hgpi;
```

Returns the current geometric line-width attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

> *-4* Error
> *>=-1* OK

Principal errors:

-

GpiSetLineEnd

```
BOOL GpiSetLineEnd (hgpi, line_end)
HPS hgpi;
LONG line_end;
```

Sets the current line-end attribute to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

line_ end

Valid values for *line_ end* are:

```
LINEEND_DEFAULT     (0) - Use default
LINEEND_FLAT        (1) - Flat
LINEEND_SQUARE      (2) - Square
LINEEND_ROUND       (3) - Round
```

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_LINE_END_ATTR
```

GpiQueryLineEnd

```
LONG GpiQueryLineEnd (hgpi)
HPS hgpi;
```

Returns the current line-end attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Line end

Principal errors:

GpiSetLineJoin

```
BOOL GpiSetLineJoin (hgpi, line_join)
HPS hgpi;
LONG line_join;
```

Sets the current line-join attribute to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

line_join

Valid values for *line_join* are:

```
LINEJOIN_DEFAULT     (0) - Use default
LINEJOIN_BEVEL       (1) - Bevel
LINEJOIN_ROUND       (2) - Round
LINEJOIN_MITRE       (3) - Mitre
```

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_LINE_JOIN_ATTR
```

GpiQueryLineJoin

```
LONG GpiQueryLineJoin (hgpi)
HPS hgpi;
```

Returns the current line-join attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

```
-1 Error
>=0 Line join
```

Principal errors:

GpiSetLinePatternSet

```
BOOL GpiSetLinePatternSet (hgpi, line_pattern_set)
HPS hgpi;
LONG line_pattern_set;
```

Sets the current line-pattern-set attribute to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

line_pattern_set
          Valid values for *line_pattern_set* are:

> *0* Use default
>
> *64* Range of values for
> *thru* the ID of a loaded
> *239* symbol set.
>
> *240* Base pattern set.

Returns:

> *0* Error
> *1* OK

Principal errors:

GPIERR_INVALID_LINE_PATTERN_SET_ATTR

GpiQueryLinePatternSet

```
LONG GpiQueryLinePatternSet (hgpi)
HPS hgpi;
```

Returns the current line-pattern-set attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

> *-1* Error
> *>=0* Line pattern set

Principal errors:

–

GpiSetLinePatternSymbol

```
BOOL GpiSetLinePatternSymbol (hgpi, line_pattern_symbol)
HPS hgpi;
LONG line_pattern_symbol;
```

Sets the current line-pattern-symbol attribute to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

line_ pattern_ symbol
Valid values for *line_ pattern_ symbol* with the base pattern set are:

```
BASESYM_SHADE1          ( 1) - Hi intensity shading
BASESYM_SHADE2          ( 2) - )
BASESYM_SHADE3          ( 3) - )
BASESYM_SHADE4          ( 4) - )- Solid shading with
BASESYM_SHADE5          ( 5) - )   decreasing intensity
BASESYM_SHADE6          ( 6) - )
BASESYM_SHADE7          ( 7) - )
BASESYM_SHADE8          ( 8) - Lo intensity shading
BASESYM_VERT            ( 9) - Vertical lines
BASESYM_HORIZ           (10) - Horizontal lines
BASESYM_DIAGUP1         (11) - Bottom left to top right
BASESYM_DIAGUP2         (12) - Bottom left to top right
BASESYM_DIAGDOWN1       (13) - Top left to bottom right
BASESYM_DIAGDOWN2       (14) - Top left to bottom right
BASESYM_NOSHADING       (15) - No shading
BASESYM_SOLID           (16) - Solid color

BASESYM_BLANK           (64) - Blank
```

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_LINE_PATTERN_ATTR

GpiQueryLinePatternSymbol

```
LONG GpiQueryLinePatternSymbol (hgpi)
HPS hgpi;
```

Returns the current line-pattern-symbol attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Line pattern symbol

Principal errors:

-

7.1.18.3 Primitive Functions

GpiSetCurrentPosition

```
BOOL GpiSetCurrentPosition (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

Sets the current x,y position to the specified value.

This function also has the effect of resetting position within a line-type sequence, and, if within an area, of starting a new closed figure and causing any previous one to be auto-closed if necessary.

The only difference between this function and GpiMove is that with this function the value of the current position may be pushed onto the attribute stack before changing it if the attribute mode (see GpiSetAttrMode)is set appropriately.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi   The handle for the GPI presentation space.

x,y   The co-ordinates of the point to which current position is to be moved.

Returns:

0 Error
1 OK

Principal errors:

-

GpiQueryCurrentPosition

```
BOOL GpiQueryCurrentPosition (hgpi, x, y)
HPS hgpi;
LONG *x;
LONG *y;
```

Queries the current x,y position.

This function is only meaningful in draw or draw-and-store modes.

Parameters:

hgpi   The handle for the GPI presentation space.

*x,*y   Variables in which current position is returned.

Returns:

```
0 Error
1 OK
```

Principal errors:

-

GpiMove

```
BOOL GpiMove (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

Sets the current x,y position to the specified value.

This function also has the effect of resetting position within a line-type sequence, and, if within an area, of starting a new closed figure and causing any previous one to be auto-closed if necessary.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi   The handle for the GPI presentation space.

x,y   The co-ordinates of the point to which current position is to be moved.

Returns:

```
0 Error
1 OK
```

Principal errors:

-

GpiLine

```
SHORT GpiLine (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

Draws a straight line from current position to the specified end-point.

Upon completion, the current x,y position is at the specified end point.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi The handle for the GPI presentation space.

x,y The co-ordinates of the end-point.

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

-

GpiPolyLine

```
SHORT GpiPolyLine (hgpi, n, x, y)
HPS hgpi;
LONG n;
LONG x[];
LONG y[];
```

Draws a series of one or more connected lines, starting at current position.

Upon completion, the current x,y position is the end point of the last line in the series.

The maximum number of lines allowed in the polyline depends upon the length of co-ordinates but is at least 8000.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi The handle for the GPI presentation space.

n   The number of x,y pairs.

x   An array of integer values (x coordinates).

y   An array of integer values (y coordinates).

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

GPIERR_INVALID_ARRAY_COUNT

for example

GpiBox

```
SHORT GpiBox (hgpi, control, x1, y1, h_round, v_round)
HPS hgpi;
LONG control;
LONG x1;
LONG y1;
LONG h_round;
LONG v_round;
```

Draws a rectangular box with one corner at current position and the diagonally opposite corner at the specified position. The sides of the box are parallel to the x and y axes, in Drawing Order Co-ordinate Space.

The corners of the box may be rounded by means of quarter ellipses, with the specified horizontal and vertical axis lengths.

The box is defined to be drawn from (x0,y0) to (x1,y0) to (x1,y1) to (x0,y1) to (x0,y0). This is relevant to, for example, area winding mode - see GpiBeginArea in the section, "Area Functions".

Current position is unchanged by this function.

Either the outline of the box, or its interior, or both, may be drawn. If the interior is to be drawn, then this function must not itself be within a GpiBeginArea - GpiEndArea bracket. In this case also, a correlation hit will result if the pick window is wholly within the box; if the outline only is drawn, the pick window must intersect the outline to score a hit.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi   The handle for the GPI presentation space.

control Specifies whether the interior of the box should be filled, and whether the outline should be drawn, as follows:-

1 Fill interior
2 Draw outline
3 Draw outline and fill interior

x1,y1 The co-ordinates of the diagonally opposite corner

h_round, v_round
Specify the horizontal and vertical length of the *full* axes of the ellipses used for rounding at each corner. If either of these is zero, no rounding occurs.

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

GPIERR_INVALID_BOX_CONTROL
GPIERR_INVALID_ROUNDING_PARAMETERS

GpiBeginStrokes

```
SHORT GpiBeginStrokes (hgpi)
HPS hgpi;
```

Defines the start of a strokes bracket. The significance of a strokes bracket is that if geometric thick lines are in force (see GpiSetLineWidthGeom), the primitives within the bracket will be drawn as a whole.

For example, enclosing the appropriate primitives within a strokes bracket will achieve the following:-

1. A line may be joined to, for example, an arc. The common point will be handled according to the Line Join parameter, rather than applying Line Ends at each end.

2. If the end of a polyline is coincident with the start, then the joining rules will be followed rather than the ending rules, at the start/end point.

3. A series of lines in XOR mix may be drawn so that intersections are not XOR'ed out. The series may be longer than just a single polyline or polyfillet.

Any geometric thick lines not drawn within a bracket have line ends at each end of the primitive.

The strokes bracket is terminated by GpiEndStrokes. This must occur in the same segment as the GpiBeginStrokes. The bracket may occur within an area bracket, but cannot cross an area bracket boundary. If within an area bracket, closure lines will

be generated as normal; otherwise, closure lines are not generated.

It is feasible for move functions to occur within the bracket, for example an 'X' in thick lines.

The following are the only primitive/attribute functions allowed to the same GPI presentation space within a strokes bracket:-

```
GpiBeginElement (containing only valid function(s))
GpiElement (containing a valid function)
GpiEndElement
GpiSetModelTransform
GpiCallSegment
GpiSetAttrMode
GpiQueryAttrMode
GpiPop (Providing only a valid function is popped)
GpiSetCurrentPosition
GpiQueryCurrentPosition
GpiMove
GpiLine
GpiPolyLine
GpiBox
GpiSetArcParams
GpiQueryArcParams
GpiArc
GpiFullArc
GpiPartialArc
GpiPieSlice
GpiPolySpline
GpiPolyFillet
GpiPolyFilletSharp
```

Parameters:-

hgpi   The handle for the GPI presentation space

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

GPIERR_ATTEMPT_TO_START_SECOND_STROKES_BRACKET

GpiEndStrokes

```
SHORT GpiEndStrokes (hgpi)
HPS hgpi;
```

Terminates a strokes bracket (see GpiBeginStrokes).

Parameters:-

hgpi   The handle for the GPI presentation space

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

```
GPIERR_END_STROKES_IGNORED
```

7.1.18.4   Visibility Functions

GpiPtVisible

```
SHORT GpiPtVisible (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

This checks whether a point is visible within the clipping region
of the device associated with the specified presentation space.
The clipping region for this purpose is defined as the intersection
between the application clipping region, and any clipping as a
result of windowing etc.

Parameters:

hgpi   The handle of a Gpi presentation space.

x,y   The co-ordinates of the point in world co-ordinates.

Returns:

```
0 Error
1 Not visible
2 Visible
```

Principal errors:

-

GpiRectVisible

```
SHORT GpiRectVisible (hgpi, x1, y1, x2, y2)
HPS hgpi;
LONG x1;
LONG y1;
LONG x2;
LONG y2;
```

This checks whether any part of a rectangle lies within the clip-
ping region of the device associated with the specified presenta-
tion space. The clipping region for this purpose is defined as the
intersection between the application clipping region, and any

clipping as a result of windowing etc.

Parameters:

hgpi   The handle of a Gpi presentation space.

x1,y1   The co-ordinates of the bottom left corner of the rectangle in world co-ordinates.

x2,y2   The co-ordinates of the top right corner of the rectangle in world co-ordinates.

Returns:

```
0 Error
1 Not visible
2 Some of the rectangle is visible
3 All of the rectangle is visible
```

Principal errors:

-

7.1.19   Arc Functions

Functions described in this section are for drawing arcs, including

- 3-point arcs
- Full arcs
- Partial arcs
- Fillets
- Fillets with specified sharpness
- Splines

The shape of the arc drawn depends upon the *arc parameters*. This can be set, and the value pushed, onto the segment call stack, just like attributes.

The attributes which control the color, width, etc, of the arc are the line attributes, described in the section, "Line Functions".

7.1.19.1   Attribute Setting Functions

GpiSetArcParams

```
BOOL GpiSetArcParams (hgpi, p, q, r, s)
HPS hgpi;
LONG p;
LONG q;
LONG r;
```

LONG s;

Sets the current arc parameters to the specified values.

The arc parameters define the shape and orientation of an ellipse which is used for subsequent GpiArc, GpiFullArc, and GpiPartialArc functions. For all of these functions except GpiArc, they also determine the direction of drawing, as follows:-

- If p.q > r.s the direction is anti-clockwise

- If p.q < r.s the direction is clockwise

- If p.q = r.s a straight line is drawn

Also except for GpiArc, they define the nominal size of the ellipse, although this may be changed by using the multiplier. For GpiArc, the size of the ellipse is determined by the three points specified on GpiArc.

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 + s.q = 0, then the transform is termed orthogonal, and the line from the origin (0,0) to the point (p,s) is either the radius of the circle, or half the major axis of the ellipse. The line from the origin to the point (r,q) is either the radius of the circle, or half of the minor axis of the ellipse.

For maximum accuracy orthogonal transforms should be used.

The default values of arc parameters are

```
p = 1     r = 0
s = 0     q = 1
```

so that the figure remains a unit circle.

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.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi	The handle for the GPI presentation space.
p,s	The x,y co-ordinates of the end of the major axis, relative to 0,0.

r,q   The x,y co-ordinates of the end of the minor axis, relative to 0,0.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_ARC_PARAMETER

GpiQueryArcParams

```
BOOL GpiQueryArcParams (hgpi, p, q, r, s)
HPS hgpi;
LONG *p;
LONG *q;
LONG *r;
LONG *s;
```

This returns the current arc parameters used to draw arcs, circles, or ellipses.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*p,*s   Variables which are set to contain p and s.

*q,*r   Variables which are set to contain q and r.

Returns:

0 Error
1 OK

Principal errors:

-

7.1.19.2 Primitive Functions

GpiArc

```
SHORT GpiArc (hgpi, x1, y1, x2, y2)
HPS hgpi;
LONG x1;
LONG y1;
LONG x2;
LONG y2;
```

Creates an arc, using the current arc parameters, through three x,y positions starting at the current x,y position. The parameters specify co-ordinates that identify the second and third

positions (*x1,y1* and *x2,y2*) of the arc.

Upon completion, the current x,y position is x2,y2.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
x1,y1	The intermediate point
x2,y2	The end point

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

-

GpiFullArc

```
SHORT GpiFullArc (hgpi, control, m)
HPS hgpi;
LONG control;
LONG m;
```

Creates a full arc with its center at the current x,y position.

The current x,y position is not changed.

The arc parameters determine whether the full arc is drawn clockwise or anti-clockwise.

Either the outline of the full arc, or its interior, or both, may be drawn. If the interior is to be drawn, then this function must not itself be within a GpiBeginArea - GpiEndArea bracket. In this case also, a correlation hit will result if the pick window is wholly within the full arc; if the outline only is drawn, the pick window must intersect the outline to score a hit.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
control	Specifies whether the interior of the full arc should be filled, and whether the outline should be drawn, as follows:-

1 Fill interior
2 Draw outline
3 Draw outline and fill interior

m   A multiplier that determines the size of the arc in rela-
     tion 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 por-
     tion and the low-order word as the fractional portion.
     Thus, a value of 65536 specifies a multiplier of 1.
     There is a current implementation limit of 255 for the
     multiplier (ie a value passed of 255 x 65536 =
     16711680).

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

GPIERR_INVALID_ARC_CONTROL
GPIERR_INVALID_MULTIPLIER

GpiPartialArc

```
SHORT GpiPartialArc (hgpi, xc, yc, m, ts, te)
HPS hgpi;
LONG xc;
LONG yc;
LONG m;
LONG ts;
LONG te;
```

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 primi-
tive is defined by the parameters ts and 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. ts and 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 anti-clockwise is determined by the arc parameters,
and ts and te.

Current position is updated to the final point on the arc. Note
this difference from GpiFullArc, where current position remains
at the center of the figure. A primitive (eg GpiLine) following

GpiPartialArc will draw from the end point of the arc.

A segment of a pie may be drawn by

1. GpiMove, to center of pie
2. GpiPartialArc, drawing one spoke and the arc
3. GpiLine, back to center

The third step may be performed implicitly by autoclosure if an area is being drawn.

A closed figure bounded by a chord and an arc may be drawn by

1. GpiSetLineType to invisible
2. GpiPartialArc, with ts = te = angle2 to define the start point
3. GpiSetLineType to visible
4. GpiPartialArc, with ts = angle2, te = angle1

(If an area is to be drawn, it should be started after step 2 or 3.)

This functions can occur within the picture; for example if either is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
xc,yc	The centre point
m	A 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. There is a current implementation limit of 255 for the multiplier (ie a value passed of 255 x 65536 = 16711680).
ts,te	The start and ending angles in scaled radians, such that 2*pi is represented as $2**31-1$, as described above. If ts = te, a null arc is drawn. The values must be positive.

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

GPIERR_INVALID_MULTIPLIER
GPIERR_INVALID_ANGLE_PARAMETER

GpiPolyFillet

```
SHORT GpiPolyFillet (hgpi, n, x, y)
HPS hgpi;
LONG n;
LONG x[];
LONG y[];
```

Creates a fillet on a series of connected lines, with the first line starting at current position.

If only two points are supplied, an imaginary line is drawn from current position to the first point, and a second line from the first point to the second. A curve is then constructed, starting at current position and tangential to the first line at that point. The curve is drawn such that it reaches the last point at a tangent to the second line.

The curve has the appearance of a fillet. The lines are imaginary, and are not drawn.

If more than two points are supplied, an imaginary series of lines is constructed through them (as in GpiPolyLine). All the lines except the first and last are then divided in two at their mid-points. A series of curved fillets are then drawn, each starting at the end point of the last, at one of the mid-points.

Upon completion, the current x,y position is the end point of the last line in the series.

The maximum number of fillets allowed depends upon the length of co-ordinates but is at least 4000.

This function can occur within the picture; for example if either is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
n	The number of x,y pairs.
x[n]	An array of integer values (x coordinates).
y[n]	An array of integer values (y coordinates).

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

GPIERR_INVALID_ARRAY_COUNT

GpiPolyFilletSharp

```
SHORT GpiPolyFilletSharp (hgpi, n, x, y, s)
HPS hgpi;
LONG n;
LONG x[];
LONG y[];
LONG s[];
```

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 GpiPolyFillet, except that instead of allowing the implementation to choose the sharpness of each of the constituent fillets, these are specified explicitly by the application.

The sharpness of each fillet is defined as follows. Let A and C be the start and end points, respectively, of the fillet, and let B be the control point. Let W be the mid-point of AC. Let D be the point where the fillet intersects WB. Then the sharpness is given by

```
        sharpness = WD/DB
```

so that

```
> 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 end point of the last line in the series.

The maximum number of fillets allowed depends upon the length of co-ordinates but is at least 4000.

This function can occur within the picture; for example if either is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
n	The number of x,y pairs.
x[n]	An array of integer values (x coordinates).
y[n]	An array of integer values (y coordinates).
s	An array of (n-1) integer values of sharpness. Each value, when divided by 65536, gives the sharpness of successive fillets, as above.

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

GPIERR_INVALID_ARRAY_COUNT
GPIERR_INVALID_SHARPNESS_PARAMETER

GpiPolySpline

```
SHORT GpiPolySpline (hgpi, n, x, y)
HPS hgpi;
LONG n;
LONG x[];
LONG y[];
```

Creates a succession of Bezier splines. The first one starts from current position and goes to the third specified point, with the first and second points used as control points. Subsequent splines start from the ending point of the previous spline, and end at the next specified point but two, with the intervening points their first and second control points. It is the application's responsibility to ensure that the gradient is continuous at each end/start point, if this is required.

Upon completion, the current x,y position is the end point of the last line in the series.

The maximum number of splines allowed depends upon the length of co-ordinates but is at least 2500.

This function can occur within the picture; for example if either is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
n	The number of x,y pairs. This must be 3*s, where s is the number of splines
x[n]	An array of integer values (x coordinates).

The x and y values are each given in the following order:-

```
      c11, c12, e1, c21, c22, e2, c31, c32, e3, ...
where
      csi      is the i'th control point of
               the s'th spline
      es       is the endpoint of the s'th
               spline
```

y[n]	An array of integer values (y coordinates).

Returns:

0 `Error`
1 `OK`
2 `Correlate hit(s)`

Principal errors:

`GPIERR_INVALID_ARRAY_COUNT`

7.1.20 Area Functions

Functions described in this section are for specifying the start and end of a filled area, and for controlling the attributes within which the area is drawn.

The boundaries of an area are specified by line and/or arc primitives (see the section, "Line Functions", and the section, "Arc Functions").

An area will consist of one or more closed figures. Each of these may be specified as a single closed primitive (box or full arc), or a series of connected strokes (lines, arcs, partial arcs, or fillets).

The following functions are valid within an area:-

- GpiMove, GpiSetCurrentPosition
- GpiLine, GpiPolyLine, GpiBox
- GpiSetAttrMode, GpiSetAttrs (for line attributes)
- GpiSetColor, GpiSetBackColor
- GpiSetMix, GpiSetBackMix
- GpiSetLineType, GpiSetLineWidth, GpiSetLineWidthGeom
- GpiSetLineEnd, GpiSetLineJoin
- GpiSetLinePatternSet, GpiSetLinePatternSymbol
- GpiSetArcParams
- GpiArc, GpiFullArc, GpiPartialArc
- GpiPolyFillet, GpiPolyFilletSharp, GpiPolySpline
- GpiCallSegment
- GpiSetModelTransform
- GpiPop which pops one of the above functions
- GpiSetCharSet, GpiSetCharBox
- GpiSetCharAngle, GpiSetCharShear
- GpiSetCharDirection, GpiSetCharMode

- GpiSetCharSpacing, GpiSetCharExtra, GpiSetCharBreakExtra
- GpiSetTextAlignment
- GpiCharString, GpiCharStringAt
- GpiCharStringPos, GpiCharStringPosAt

GpiBox, GpiFullArc, and GpiPartialArc are only valid within an area
bracket (ie between GpiBeginArea and GpiEndArea) with *control* set to *2*.
Other values of *control* on these functions cause an implicit area bracket
around the function.

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

The start of a new figure is indicated by:-
- GpiMove
- GpiSetCurrentPosition
- GpiCallSegment
- GpiSetModelTransform
- GpiPop (or end of called segment) which pops current position or a
  model transform

The system ensures that each figure is closed, by adding a line from the
last point specified to the starting point of the figure if necessary.

Optionally, as well as filling the interior, the boundary lines may be
drawn; if they are, the line attributes define the appearance of these lines.

Areas are drawn with the following attributes:-
- Pattern color
- Pattern background color
- Pattern mix
- Pattern background mix
- Pattern set
- Pattern symbol
- Pattern reference point

The manipulation of the color, background color, mix, and background
mix attributes globally is described in the section, "Color and Mix Func-
tions".

The default values of pattern attributes are as follows:-

```
Pattern color:              Color 7
Pattern background color:   Color O
Pattern mix:                Overpaint
Pattern background mix:     Leave alone
Pattern set:                Standard 17-value set
Pattern symbol:             Solid
Pattern reference point:    (O,O)
```

The pattern used for area shading may either be taken from a symbol within a symbol set (mono- or multi-colored), or else it may be a bitmap.

Pattern attributes are also used for various raster operations - see the section, "Bitmap Support". They are also used when drawing the interior of a box or full arc.

7.1.20.1   Resources and Defaults Functions

GpiSetBitmapId

```
BOOL GpiSetBitmapId (hgpi, hbm, lcid)
HPS hgpi;
HBITMAP hbm;
LONG lcid;
```

Tags a bitmap with a local id, so that it may be used as a pattern set.

In order to use the bitmap for area shading (or as the pattern in a GpiBitBlt operation), a GpiSetPatternSet must be issued with the specified lcid.

Parameters:

hgpi   The handle for the GPI presentation space.

hbm   The handle of the bitmap.

lcid   A local id with which the bitmap is to be tagged.

        It is an error if the lcid is already in use to refer to a font, symbol set, or bitmap.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_BITMAP_HANDLE
GPIERR_INVALID_LCID
GPIERR_LCID_ALREADY_IN_USE
```

GpiQueryBitmapHandle

```
HBITMAP GpiQueryBitmapHandle (hgpi, lcid)
HPS hgpi;
LONG lcid;
```

Returns the handle of the bitmap currently tagged with the specified local id.

A null handle is returned if a bitmap is not currently tagged with the specified lcid.

Parameters:

hgpi   The handle for the GPI presentation space.

lcid   The local id.

Returns:

```
 0 Error
!=0 Bitmap handle
```

Principal errors:

```
GPIERR_BITMAP_NOT_FOUND
GPIERR_INVALID_LCID
```

7.1.20.2 Attribute Setting Functions

GpiSetPatternSet

```
BOOL GpiSetPatternSet (hgpi, pattern_set)
HPS hgpi;
LONG pattern_set;
```

Sets the current pattern-set attribute to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

pattern_ set
         The local identifier (lcid) of the required pattern set:

         0 Default set.

         64 Range of values for
         thru the ID of a loaded

236

239 symbol set.

240 Base pattern set.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_AREA_CONTEXT_ERROR
GPIERR_INVALID_PATTERN_SET_ATTR

GpiQueryPatternSet

```
LONG GpiQueryPatternSet (hgpi)
HPS hgpi;
```

Returns the current pattern-set attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Pattern set

Principal errors:

-

GpiSetPattern

```
BOOL GpiSetPattern (hgpi, pattern_symbol)
HPS hgpi;
LONG pattern_symbol;
```

Sets the current pattern symbol attribute to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

If the current pattern set specified a bitmap (see GpiSetBitmapId and GpiSetPatternSet), then pattern_symbol must be 65 in order to use the bitmap.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

pattern_symbol

> The identity of the required pattern. A value of *0* selects the default pattern, any other value identifies a pattern in the current pattern set. Valid values in the base pattern set are:

```
BASESYM_SHADE1       ( 1) - Hi intensity shading
BASESYM_SHADE2       ( 2) - )
BASESYM_SHADE3       ( 3) - )
BASESYM_SHADE4       ( 4) - )- Solid shading with
BASESYM_SHADE5       ( 5) - )   decreasing intensity
BASESYM_SHADE6       ( 6) - )
BASESYM_SHADE7       ( 7) - )
BASESYM_SHADE8       ( 8) - Lo intensity shading
BASESYM_VERT         ( 9) - Vertical lines
BASESYM_HORIZ        (10) - Horizontal lines
BASESYM_DIAGUP1      (11) - Bottom left to top right (1)
BASESYM_DIAGUP2      (12) - Bottom left to top right (2)
BASESYM_DIAGDOWN1    (13) - Top left to bottom right (1)
BASESYM_DIAGDOWN2    (14) - Top left to bottom right (2)
BASESYM_NOSHADING    (15) - No shading
BASESYM_SOLID        (16) - Solid color

BASESYM_BLANK           (64) - Blank
```

> If the specified pattern is not valid, the default (device dependent) is used.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_AREA_CONTEXT_ERROR
GPIERR_INVALID_PATTERN_ATTR
```

GpiQueryPattern

```
LONG GpiQueryPattern (hgpi)
HPS hgpi;
```

Returns the current pattern-symbol attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

```
-1 Error
>=0 Pattern symbol
```

Principal errors:

-

GpiSetPatternRefPoint

```
BOOL GpiSetPatternRefPoint (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

Sets the current pattern reference point to the specified value.

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 a horizontal and vertical direction.

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 GpiBitBlt 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 Gpi-FloodFill.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

The default pattern reference point is (0,0).

Parameters:

hgpi   The handle for the GPI presentation space.

x,y   The x,y co-ordinates of the pattern reference point in World Co-ordinates.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_AREA_CONTEXT_ERROR

GpiQueryPatternRefPoint

```
BOOL GpiQueryPatternRefPoint (hgpi, x, y)
```

```
HPS hgpi;
LONG *x;
LONG *y;
```

This returns the current pattern reference point, used in filling areas, and in GpiFloodFill.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi	The handle for the GPI presentation space.
*x,*y	Variables which are set to contain the pattern reference point.

Returns:

```
0 Error
1 OK
```

Principal errors:

-

7.1.20.3 Primitive Functions

GpiBeginArea

```
SHORT GpiBeginArea (hgpi, control)
HPS hgpi;
ULONG control;
```

Indicates the beginning of a set of primitives that define the boundary of an area.

The area interior is constructed either in alternate mode or in winding mode. In alternate mode, whether any point is within the interior is determined by drawing an imaginary line from that point to infinity; if there is an odd number of boundary crossings, the point is inside the area, if there is an even number of crossings, it is not.

With winding mode, the directionality of the boundary lines is taken into account. Using the same imaginary line, the number of crossings is counted as with alternate mode, but now boundary lines going in one direction score plus one, and boundary lines going in the other score minus one. The original point is in the interior if the final score is non-zero.

Although the current x,y position is not changed by GpiBeginArea, it will be affected by the drawing orders in the boundary definition.

This function can occur within the picture; for example if it is

issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi   The handle for the GPI presentation space.

control   A 4-byte parameter containing 2 flags at the least significant end (bits 0 and 1). These have the following meanings:-

Bit 0		
	0	Do not draw boundary lines
	1	Draw boundary lines
Bit 1		
	0	Construct interior in *alternate* mode
	1	Construct interior in *winding* mode

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

```
GPIERR_INVALID_AREA_CONTROL
GPIERR_ATTEMPT_TO_START_SECOND_AREA
```

GpiEndArea

```
SHORT GpiEndArea (hgpi)
HPS hgpi;
```

Indicates the end of a set of primitives that define the boundary of an area.

Upon completion, the current x,y position is the last x,y position specified in the boundary definition, unless autoclosure occurred, in which case it is the starting point of the last figure.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

```
0 Error
1 OK
2 Correlate hit
3 Multiple correlate hits
```

Principal errors:

```
GPIERR_END_AREA_IGNORED
```

7.1.21 Character Functions

Functions described in this section are for drawing character strings, and for controlling the attributes with which they are drawn.

There are three basic character string drawing primitives:-

- GpiCharString, which draws a character string consisting of a number of codepoints, where the positioning of characters after the first depends upon the character and/or font attributes

- GpiCharStringPos, as GpiCharString, but additional positioning information is provided for each character

Character strings are drawn with the following attributes:-

- Character color
- Character background color
- Character mix
- Character background mix
- Character set
- Character box
- Character angle
- Character shear
- Character direction
- Character mode
- Character spacing
- Character extra
- Character break extra
- Text alignment

The manipulation of the color, background color, mix, and background mix attributes globally is described in the section, "Color and Mix Functions".

The default character attributes are:-

```
Character color:              Color 7
Character background color:   Color 0
Character mix:                Overpaint
Character background mix:     Leave alone
```

```
Character set:                    Base set
Character box:                    Device dependent
Character angle:                  No rotation
Character shear:                  No shear
Character direction:              Left-to-right
Character mode:                   String
Character spacing:                No spacing
Character extra:                  No extra spacing
Character break extra:            No break extra spacing
Text alignment:                   Standard
```

Characters may be drawn using *symbol sets.*

Characters can also be drawn using *fonts.*

- Symbol sets have the following characteristics:-

  - Always loaded from application storage.

  - Only available to the Gpi presentation space into which they are loaded.

  - Always include the actual character definitions, in either raster (image) or vector form.

  - May be proportionally spaced.

  - The set is identified only by an 8-character name. The meaning of this name is installation-dependent.

  - The only metric information available is the box size (plus any proportional spacing data).

  - Positioned by the corner of the character cell.

  Symbol sets are loaded by the GpiLoadSymbolSet function.

  In drawing using symbol sets, characters are positioned by the appropriate corner of the character box.

- Fonts have both physical and logical characteristics.

  Physical font characteristics:-

  - Always loaded from disk storage.

  - Available across the system.

  - May include the actual character definitions.

  - Include descriptive data such as facename, pitch, whether bold, etc etc.

  - Include considerable metrics data.

  - Positioned by the font baseline (for left-to-right or right-to-left directions).

  Logical font characteristics are specified by an application:-

- Descriptive and metric data of font required.

- No actual character definitions.

Physical fonts are loaded into the system by GpiLoadFonts. The logical font definition is supplied with GpiCreateLogFont.

In drawing using fonts, characters are positioned by the baseline of the characters within the box.

7.1.21.1 Font Selection

The GpiSetCharSet function selects a Character Set or Font for use in subsequent Text drawing functions. The function specifies an LCID value. This value represents either:

1. A Symbol Set loaded by GpiLoadSymbolSet.

2. A logical font definition created by GpiCreateLogFont.

For a Symbol Set, the Symbol Set that is to be used is completely defined at the time GpiSetCharSet is issued. The appearance of text strings using the Symbol Set can be changed by the Character Drawing Attributes, however. For example, if the Symbol Set is a Vector one, changing the Character Cell size changes the size at which the characters from the set are drawn.

For a Font, the situation is different. The GpiSetCharSet function call specifies the logical font required by the application. However, the selection operates in one of two ways, depending on the setting of the *Font Use* parameter defined in the GpiCreateLogFont function:

Fixed Font mode
>   where the Font to use is completely defined by the GpiSetCharSet function and can only be changed by a susequent GpiSetCharSet call.

Meta Font mode
>   where the Font to use is defined by the GpiSetCharSet call but can be modified by subsequent changes to the Character Drawing Attributes.

Fixed Font mode.

In Fixed Font mode, the font used is completely specified by the GpiCreateLogFont function. It is not affected by any subsequent setting of the Character Drawing Attributes. Using this mode, an application can set up a set of fonts in advance of drawing, knowing that the fonts will not be changed during picture drawing.

244

Meta Font mode. In Meta Font mode, GpiCreateLogFont generates a complete logical definition of a font. However, after selecting a particular font definition using GpiSetCharSet, some of the Character Drawing Attributes can subsequently change the font in use. For example, changing the Character Cell attribute will change the size of font in use.

Thus Meta Font mode implies a fresh mapping of the LCID to an actual font each time a Character Drawing Attribute is changed. In practice, the mapping will be done when the next Character String is drawn. At this point, both the LCID and the current Character Drawing Attributes define the required font, which is then chosen.

The character attributes involved in the selection of the font are:

Character Cell
> affects the point size of font selected and also any scaling that is applied when realising the font.

Character Direction
> affects the font selected - the font should be designed for the specified direction.

Character Angle
> should be satisfied by the range of baseline directions supported by the font.

Character Shear
> specifies a shear angle which should be matched by the slope range supported by the font.

Character Precision
> defines the drawing fidelity required by the user. The font should be capable of matching the fidelity requested. Precision 3 implies that the font's characters should be transformable at will, for example.

7.1.21.2 Fonts Which are Supplied with Presentation Manager

The fonts that are supplied with Presentation Manager are:

- System font

  Size appropriate to device resolution
- Times Roman

  8, 10, 12, 16, 18 point sizes
- Helvetica

  8, 10, 12, 16, 18 point sizes

- Courier

  8, 10, 12 point sizes

The listed fonts are available for all display adapters supported by Presentation Manager, and also for two printer resolutions. These sizes are supported by image fonts, to provide greatest legibility and high performance.

In addition, each typeface is also supported by an outline font, in

- Normal
- Bold
- Italic
- Bold italic

These provide for a wider range of point sizes, ... a wider range of point sizes, but with generally lower legibility and lower performance.

Note that additional fonts can be added to the system as required, but these are not supplied with Presentation Manager itself.

7.1.21.3 Resources and Defaults Functions

GpiLoadSymbolSet

```
BOOL GpiLoadSymbolSet (hgpi, name, lcid, length, data)
HPS hgpi;
LPSTR name;
LONG lcid;
LONG length;
LPBUF data;
```

Loads a symbol-set definition, from a specified application data area, into the GPI. (The data starts with the 'length' field, and the symbol set id and store number fields are ignored.)

Parameters:

---

hgpi   The handle for the GPI presentation space.

name   An 8-character name which may be used to describe the symbol set, for interchange for example.

lcid   The local identifier (lcid) for the symbol set. This must be in the range *64* through *239*.

It is an error if the lcid is already in use to refer to a font, symbol set, or bitmap; except that an incremental load is allowed to a symbol set, providing it is not currently selected.

length   The length of the data. This should be the same as the length field at the start of the data.

data   The start address of the data area which contains the symbol set definition.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_NAME
GPIERR_INVALID_LENGTH_FOR_SYMBOL_SET
GPIERR_INVALID_EXTENDED_HEADER_LENGTH
GPIERR_INCOMPLETE_SYMBOL_DEFN
GPIERR_INVALID_LCID
GPIERR_INVALID_SYMBOL_SET_TYPE
GPIERR_INCONSISTENT_SYMBOL_SET_TYPE
GPIERR_LCID_ALREADY_IN_USE
GPIERR_INVALID_SYMBOL_SET_FORMAT
GPIERR_INADEQUATE_LENGTH_FOR_SYMBOL_SET
GPIERR_INVALID_SYMBOL_SET_CODEPOINT
GPIERR_UNSUPPORTED_SYMBOL_SET_OPTION
GPIERR_CONFLICTING_SYMBOL_SET_ID
GPIERR_INVALID_VECTOR_SYMBOL_DEFN
```

GpiLoadFonts

```
BOOL GpiLoadFonts (hab, filename)
HAB hab;
LPSZ filename;
```

Loads font(s) from the specified resource file. All of the fonts in the file become available for all applications to use. The font definition is as produced by the Font Editor.

Parameters:

hab   The anchor block handle

filename
          The filename of the font resource file.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_FONT_NOT_FOUND
GPIERR_INVALID_FONT_FILENAME
GPIERR_INVALID_FONT_DEFN
Others TBD
```

GpiUnloadFonts

```
BOOL GpiUnloadFonts (hab, filename)
HAB hab;
LPSZ filename;
```

Unloads font definition(s) which had previously been loaded from the specified resource file.

Parameters:

hab   The anchor block handle

filename
     The filename of the font resource file.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_FONT_NOT_LOADED
GPIERR_INVALID_FONT_FILENAME
```

GpiCreateLogFont

```
BOOL GpiCreateLogFont (hgpi, name, lcid, attrs)
HPS hgpi;
LPSTR name;
LONG lcid;
LPBUF attrs;
```

A logical definition of a font (ie in terms of its attributes) which the application requires to use. The system will utilise whichever of the fonts at its disposal most closely matches the requirements. An application may, however, force selection of a particular font by quoting the *match* value in *attrs*, to be that returned for the desired font by GpiQueryFonts.

The local identifier (*lcid*) which the application wishes to use to reference this logical font for later drawing operations is also specified. See GpiSetCharSet.

If *facename* in *attrs* is null, and all of the attributes except the code page are set to zero, the system default font will be selected, in the specified code page.

Parameters:

hgpi   The handle for the GPI presentation space.

name   An 8-character name which may be used to describe the logical font. Its main use is interchange files, where it can help to identify the required font. For example, it can reference a filename which contains the

font for a remote system.

lcid   The local id which the application will use to refer to this font. This must be in the range *64* through *239*.

It is an error if the lcid is already in use to refer to a font, symbol set, or bitmap.

attrs   The address of a buffer containing the required attributes of the font. The format is as follows:-

```
Field Name                      Field Type
==========                      ==========

Length of record                2 byte integer
Match                           4 byte integer
Facename                        FACESIZE size string
Registry id                     2 byte integer
Code page                       2 byte integer
Height                          4 byte integer
Average character width         4 byte fixed point
Width class                     2 byte integer
Weight class                    2 byte integer
Selection flags                 2 byte flags
Type flags                      2 byte flags
Quality                         2 byte integer
Font use flags                  2 byte flags
```

For more information see Appendix A.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_FONT_NOT_LOADED
GPIERR_INVALID_LCID
GPIERR_LCID_ALREADY_IN_USE
GPIERR_INVALID_FONT_ATTRS
GPIERR_INVALID_NAME
Others TBD
```

GpiDeleteSetId

```
BOOL GpiDeleteSetId (hgpi, lcid)
HPS hgpi;
LONG lcid;
```

Does one of the following, depending upon the object referred to by *lcid* :-

- If *lcid* refers to a loaded symbol set, the symbol set is deleted, and is no longer available for use.

- If *lcid* refers to a logical font, the logical font is deleted, and is no longer available for use.

- If *lcid* refers to a bitmap, then the bitmap is no longer tagged with the lcid. The bitmap handle is still valid.

In all cases the lcid is freed and is now available for re-use.

Optionally, this operation may be performed for every lcid currently in use.

Parameters:

hgpi   The handle for the GPI presentation space.

lcid   The local identifier (lcid) for the object.

If *-1* is specified, *all* loaded symbol sets, are deleted. Also all logical fonts are deleted, and all bitmap tagging removed.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_LCID
GPIERR_LCID_NOT_LOADED

GpiQueryNumberSetIds

```
LONG GpiQueryNumberSetIds (hgpi)
HPS hgpi;
```

Returns the total number of lcids currently in use, referring to symbol sets, fonts, or bitmaps. This may be used to perform a subsequent GpiQuerySetIds.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Number of lcids in use

Principal errors:

-

GpiQuerySetIds

```
BOOL GpiQuerySetIds (hgpi, n, types, names, lcids)
HPS hgpi;
LONG n;
LONG types[];
LPSTR names[];
LONG lcids[];
```

Returns data about loaded symbol sets (see GpiLoadSymbolSet),

fonts (see GpiCreateLogFont), and tagged bitmaps (see GpiSet-BitmapId). Information about the first *n* objects is returned; if there are fewer than *n*, *types* and *lcids* elements for the remainder are set to zero.

Parameters:

hgpi	The handle for the GPI presentation space.
n	The number of objects to be queried.
	The number of lcids currently in use, and therefore the maximum number of objects for which information can be returned, may be found with GpiQueryNumber-SetIds.
types[n]	An array of length n.

1 Image symbol set
2 Vector symbol set
6 Logical font
7 Bitmap id

names[n]

An array of pointers to 8-byte fields, into which the 8-character names associated with the symbol sets are returned.

A similar name is returned for logical fonts while for bitmaps, an all-blank name is returned.

lcids[n]   An array in which the lcids are returned.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_ARRAY_COUNT

GpiQuerySymbolSetData

```
BOOL GpiQuerySymbolSetData (hgpi, lcid, count, data)
HPS hgpi;
LONG lcid;
LONG count;
LONG data[];
```

Returns data about a symbol set loaded with GpiLoadSymbolSet.

Parameters:

hgpi   The handle for the GPI presentation space.

lcid   The local identifier (lcid) for the symbol set for which information is required.

It is an error if the lcid value does not refer to a currently loaded symbol set.

count   The number of elements supplied in *data*.

data[count]
          An array of length *count*. Up to *count* elements are returned.

1	The width of the box in which the symbol is defined.
2	The height of the box in which the symbol is defined.
3	Whether the symbol set supports proportional spacing. *1* if so, otherwise *0*.

Any further elements will be zeroed.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_LCID
GPIERR_LCID_NOT_A_SYMBOL_SET
GPIERR_INVALID_ARRAY_COUNT
```

GpiQueryFonts

```
LONG GpiQueryFonts (hgpi, facename, n,
       metrics_length, metrics)
HPS hgpi;
LONG x;
LPSZ facename;
LONG *n;
LONG metrics_length;
LPBUF metrics;
```

This returns a record providing details of the fonts which match the specified *facename*.

Font *metrics* are returned for as many matching fonts as there is room for in *metrics*.

By inspecting the returned data, the application may choose which of the available fonts is most appropriate for its requirements. If necessary, it can force selection of a particular font, by specifying its *match* (as returned in *metrics*) in the *attrs* structure for GpiCreateLogFont.

By specifying *n* as *0*, and then looking at the value returned, an

application can determine how many fonts there are which match the *facename.*

All sizes are returned in world co-ordinates.

Parameters:

hgpi   The handle for the GPI presentation space.

facename
           The facename of the fonts of interest.

*n   The number of fonts for which the application requires
           the metrics. The number of fonts that were actually
           returned is returned in this variable.

metrics_length
           The length of each metrics record to be returned. The
           buffer pointed to by *metrics* must be *n multiplied by
           metrics_length long.

metrics   In this buffer are returned the font metrics of up to *n
           matching fonts. The format for each record is as
           defined for GpiQueryFontMetrics. For each font, no
           more data than *metrics_length* will be returned.

Returns:

-1 Error
>=0 Number of fonts not returned

Principal errors:

GPIERR_INVALID_FACENAME
GPIERR_INVALID_ARRAY_COUNT (METRICS)
GPIERR_INVALID_LENGTH (metrics_length)

GpiQueryFontMetrics

```
BOOL GpiQueryFontMetrics (hgpi, metrics_length, metrics)
HPS hgpi;
LONG metrics_length;
LPBUF metrics;
```

This returns a record providing details of the font metrics for
the currently selected logical font.

All sizes are returned in world co-ordinates.

Parameters:

hgpi   The handle for the GPI presentation space.

metrics_length
           The length of the buffer pointed to by *metrics.*

metrics   In this buffer are returned the font metrics of a matching font. The format is

```
Field Name                       Field Type
==========                       ==========

Family name                      FACESIZE size string
Facename                         FACESIZE size string
Registry id                      2 byte integer
Code page                        2 byte integer
Em height                        2 byte integer
X height                         2 byte integer
Max ascender                     2 byte integer
Max descender                    2 byte integer
Lower case ascent                2 byte integer
Lower case descent               2 byte integer
Internal leading                 2 byte integer
External leading                 2 byte integer
Average character width          2 byte integer
Maximum character increment      2 byte integer
Maximum baseline extent          2 byte integer
Character slope                  2 byte integer
Inline direction                 2 byte integer
Character rotation               2 byte integer
Weight class                     2 byte integer
Width class                      2 byte integer
X device resolution              2 byte integer
Y device resolution              2 byte integer
First character                  1 byte unsigned
Last character                   1 byte unsigned
Default character                1 byte unsigned
Break character                  1 byte unsigned
Nominal point size               2 byte integer
Minimum point size               2 byte integer
Maximum point size               2 byte integer
Type flags                       2 byte flags
Selection flags                  2 byte flags
Capabilities                     2 byte flags
Subscript size                   2 byte integer
Subscript position               2 byte integer
Superscript size                 2 byte integer
Superscript position             2 byte integer
Underscore width                 2 byte integer
Underscore spacing               2 byte integer
Strikeout size                   2 byte integer
Strikeout position               2 byte integer
Kerning pairs                    2 byte integer
Kerning tracks                   2 byte integer
Match                            4 byte integer
```

No more data than *metrics_length* will be returned.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_LENGTH (metrics_length)

GpiQueryKerningPairs

```
BOOL GpiQueryKerningPairs (hgpi, count, data)
HPS hgpi;
LONG count;
LPBUF data;
```

This returns kerning pair information for the currently selected logical font.

Note that the number of kerned pairs is a field in the text metrics.

Parameters:

hgpi	The handle for the GPI presentation space.
count	The number of kerning pairs for which there is room in *data*.
data	An array of *count* kerning pair records in which information is returned. No more than *count* records will be returned.

Each record consists of the following:-

```
Field Name                    Field Type
==========                    ==========

First character of pair       2 byte integer
Second character of pair      2 byte integer
Kerning amount                2 byte signed
                     integer, with positive
                     numbers indicating
                     increased inter-character
                     spacing
```

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_ARRAY_COUNT

GpiQueryKerningTracks

```
BOOL GpiQueryKerningTracks (hgpi, count, data)
HPS hgpi;
LONG count;
LONG data[];
```

This returns kerning track information for the currently selected logical font.

Parameters:

hgpi   The handle for the GPI presentation space.

count   The number of elements contained in *data*. The number which should be allowed for in order to retrieve the full kerning track data for this font may be found by GpiQueryFontMetrics.

data[count]
        An array of *count* elements, in which kerning track information is returned. No more than *count* elements will be returned.

Returns:

 0 Error
 1 OK

Principal errors:

GPIERR_INVALID_ARRAY_COUNT

GpiQueryWidthTable

```
BOOL GpiQueryWidthTable (hgpi, first_char, count, data)
HPS hgpi;
LONG first_char;
LONG count;
LONG data[];
```

This returns width table information for the currently selected logical font.

Parameters:

hgpi   The handle for the GPI presentation space.

first_char
        The codepoint of the initial character for which width table information is required.

count   The number of elements contained in *data* (i.e. the number of elements to be returned). The number which should be allowed in order to retrieve the the full width table data for this font may be found by GpiQueryFontMetrics.

data[count]
        An array of *count* elements, in which width table information is returned. No more than *count* elements will be returned.

Returns:

 0 Error
 1 OK

Principal errors:

```
GPIERR_INVALID_FIRST_CHAR
GPIERR_INVALID_ARRAY_COUNT
```

GpiEnableKerning

```
BOOL GpiEnableKerning (hgpi, flags)
HPS hgpi;
ULONG flags;
```

This enables or disables pair and track kerning. Note that this applies globally for the presentation space, not individually by font.

By default, both pair and track kerning are on. The kerning state is not metafiled, but is retained across re-associations.

It is an error to issue this function in any of the following cases:-

- Inside an open segment
- Outside an open segment, but inside one of the following:-
    - Area bracket
    - Strokes bracket
    - Element bracket
    - Clip area bracket

Parameters:

hgpi	The handle for the GPI presentation space.
flags	Option flags, as follows:-

EK_PAIR (Bit 0)
> Set to B'1' to enable pair kerning, B'0' to disable pair kerning

EK_TRACK (Bit 1)
> Set to B'1' to enable track kerning, B'0' to disable track kerning

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_KERNING_FLAGS
```

GpiQueryKerning

```
LONG GpiQueryKerning (hgpi)
HPS hgpi;
```

This returns whether pair and track kerning are currently enabled.

The return parameter is a set of flags, as follows:-

EK_PAIR (Bit 0)
> Set to B'1' if pair kerning is enabled, B'0' otherwise

EK_TRACK (Bit 1)
> Set to B'1' if track kerning is enabled, B'0' otherwise

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Flags

Principal errors:

-

GpiSetCp

```
BOOL GpiSetCp (hgpi, codepage)
HPS hgpi;
LONG codepage;
```

This selects the codepage id to be used for graphics characters for the base (default) character set.

Any one of three codepages may be selected, either of the two ASCII codepages defined to DOS, or the additional EBCDIC (CECP) one appropriate to the national language version of shipment.

When a Gpi presentation space is first created, the code page in force is that defined by the process code page.

Parameters:

hgpi   specifies the handle for the GPI presentation space.

codepage
> The codepage id. WinQueryCpList can be used to find which ones are available.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_CODEPAGE

GpiQueryCp

```
LONG GpiQueryCp (hgpi)
HPS hgpi;
```

This returns the currently selected graphics codepage id.

Parameters:

hgpi   specifies the handle for the GPI presentation space.

Returns:

0 Error
!=0 Code page

Principal errors:

-

GpiQueryTextBox

```
BOOL GpiQueryTextBox (hgpi, n, ch, count, x, y)
HPS hgpi;
LONG n;
LPSTR ch;
LONG count;
LONG *x;
LONG *y;
```

This processes the specified string as if it were to be drawn, under the current character attributes, and returns an array of up to 5 (x,y) co-ordinate pairs. The first four of these pairs are the co-ordinates 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, which is the position at which a subsequent string would have to be drawn if it were to follow on smoothly.

All co-ordinates are relative to the start point of the string, as defined by the character direction.

This information may be used to box or underline the string, or to change the attributes in the middle of a longer string.

Parameters:

hgpi   The handle for the GPI presentation space.

n   The length of the string

ch   The character string to be examined

count   The number of elements in *x and *y.

*x,*y   Variables in which the co-ordinate pairs are returned.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_DBCS_SYMBOL_SET_NOT_AVAILABLE (warning)
GPIERR_INVALID_CHAR_STRING_LENGTH
GPIERR_INVALID_DBCS_CHAR_IN_STRING (warning)
GPIERR_DBCS_CHAR_STRING_MUST_HAVE_EVEN_LENGTH (warning)
GPIERR_INVALID_ARRAY_COUNT
```

GpiQueryTextBreak

```
LONG GpiQueryTextBreak (hgpi, n, ch, extent, count)
HPS hgpi;
LONG n;
LPSTR ch;
LONG extent;
LONG *count;
```

This processes the specified string as if it were to be drawn,
under the current character attributes, and finds where the
string must be split if it is not to exceed the specified extent.

By breaking the string at the indicated character (or before),
and distributing the remainder amongst the characters using
GpiSetTextCharacterExtra, text justification may be achieved.

Parameters:

hgpi	The handle for the GPI presentation space.
n	The length of the string
ch	The character string to be examined
extent	The maximum extent for the string, measured along the baseline for left-to-right and right-to-left characters directions, and along the shearline for top-to-bottom and bottom-to-top character directions.
*count	A variable in which the number of characters which fit within the extent is returned. If no characters fit, zero is returned.

Returns:

-1 Error
>=0 Amount of *extent* which will be unused if only
    count characters are kept in the string

Principal errors:

```
GPIERR_DBCS_SYMBOL_SET_NOT_AVAILABLE (warning)
GPIERR_INVALID_CHAR_STRING_LENGTH
GPIERR_INVALID_DBCS_CHAR_FOUND_IN_STRING (warning)
GPIERR_DBCS_CHAR_STRING_MUST_HAVE_EVEN_LENGTH (warning)
GPIERR_INVALID_EXTENT
```

GpiQueryCharCorr

```
BOOL GpiQueryCharCorr (hgpi)
HPS hgpi;
```

This function returns an offset which indicates which character within a string was selected, the last time that a character string primitive returned a successful correlation hit. It is valid whether the correlation was against a primitive stored in a segment, or processed in draw or draw-and-store mode.

If more than one character in the string was selected, the offset of the first one in the string is returned.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

```
-4 Error
-1 No offset
>=0 Ordinal number of character (first is 0) within the string
```

Principal errors:

–

GpiQueryDefCharBox

```
BOOL GpiQueryDefCharBox (hgpi, width, height)
HPS hgpi;
LONG *width;
LONG *height;
```

This function returns the size of the default graphics character box, in world co-ordinates.

Parameters:

hgpi   The handle for the GPI presentation space.

*width, *height
     Variables in which the default width and height are returned.

Returns:

```
0 Error
1 OK
```

Principal errors:

–

7.1.21.4   Attribute Setting Functions

GpiSetCharSet

```
BOOL GpiSetCharSet (hgpi, set)
HPS hgpi;
LONG set;
```

Sets the current character-set attribute to the specified value.

This functions can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

set   The local identifier (lcid) of the required character set.

```
0 Default set
1 APL set (if featured; character mode 1 only)
8 Default DBCS character set (if featured)

64  Range of values for
thru the ID of a loaded
239 logical font or symbol set.

240 Base character set.
```

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_CHAR_SET_ATTR
```

GpiQueryCharSet

```
LONG GpiQueryCharSet (hgpi)
HPS hgpi;
```

Returns the current character set attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

```
-1 Error
>=0 Character set (lcid)
```

Principal errors:

-

GpiSetCharBox

```
BOOL GpiSetCharBox (hgpi, width, height)
HPS hgpi;
LONG width;
LONG height;
```

Sets the current character-box attribute to the specified value.

The parameters *width* and *height* specify values for the width and height of a character box in terms of the drawing order coordinate space.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

width,height
          The width and height of the character box. These are long signed integers, with a notional binary point between the second and third bytes. Thus a width of 8 world co-ordinate units is represented as 8*65536.

          The width determines the spacing of consecutive characters along the baseline.

          Both width and height can be positive, negative, or zero.

          When either parameter is negative, the spacing occurs in the opposite direction to normal *and each character is drawn reflected* in character mode 3. Thus, for example, a negative height in the standard direction in mode 3 means that the characters are drawn upside down, and the string drawn below the baseline (assuming no other transformations cause inversion).

          A zero character width or height is also valid; here, the string of characters collapses into a line. If both are zero, the string is drawn as a single point.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_CHAR_BOX_ATTR

GpiQueryCharBox

```
BOOL GpiQueryCharBox (hgpi, width, height)
HPS hgpi;
LONG *width;
LONG *height;
```

Returns the current character box attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*width,*height
          Variables in which the character box attribute are
          returned.

Returns:

0 Error
1 OK

Principal errors:

-

GpiSetCharAngle

```
BOOL GpiSetCharAngle (hgpi, ax, ay)
HPS hgpi;
LONG ax;
LONG ay;
```

Sets the current character-angle attribute to the specified value.
The parameters *ax* and *ay* specify integer values for the coordi-
nates of the end of a line starting at the origin (0,0); the base
line for subsequent character strings is parallel to this line.

In character-mode 1, the call has no effect when characters are
drawn.

In character-mode 2, the angle is used to determine the position
of each character, but the orientation of characters within the
character box is inherent in their definitions. The characters are
positioned so that the lower left-hand corners of the character
definitions are placed at the lower left-hand corners of the char-
acter boxes.

In character-mode 3, the angle is observed accurately, and the
character boxes are rotated to be normal to the character base-
line. If the co-ordinate system is such that one x-axis unit is not
physically equal to one y-axis unit, a rotated character string
appears to be sheared.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi	The handle for the GPI presentation space.
ax,ay	The co-ordinates of a point defining the baseline. If both *ax* and *ay* are zero, the character angle is reset to the default value.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_CHAR_ANGLE_ATTR
```

GpiQueryCharAngle

```
BOOL GpiQueryCharAngle (hgpi, ax, ay)
HPS hgpi;
LONG *ax;
LONG *ay;
```

Returns the current character angle attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi	The handle for the GPI presentation space.
*ax,*ay	Variables in which the character angle attribute are returned.

Returns:

```
0 Error
1 OK
```

Principal errors:

-

GpiSetCharShear

```
BOOL GpiSetCharShear (hgpi, hx, hy)
HPS hgpi;
LONG hx;
LONG hy;
```

Sets the current character-shear attribute to the specified value.

The parameters *hx* and *hy* specify integer values that identify the end coordinates of a line originating at 0,0; the vertical strokes in subsequent character strings are drawn parallel to the defined line. The top of the character box remains parallel to the character baseline.

If *hx = 0* and *hy = 1* (the default), 'upright' characters result. If *hx* and *hy* are both positive or both negative, the characters slope from bottom left to top right. If *hx* and *hy* are of opposite signs, the characters slope from top left to bottom right. No character inversion takes place as a result of shear alone. (Inversion can be performed with the GpiCharBox call).

Usually, it is an error to specify a zero value for *hy*, because this would imply an infinite shear. However, if both *hx* and *hy* are zero, the attribute is set to the default value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

hx,hy   The co-ordinates of a point defining the shearline.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_CHAR_SHEAR_ATTR

GpiQueryCharShear

GpiQueryCharShear (hgpi, hx, hy)
        HPS hgpi;
        LONG *hx;
        LONG *hy;

Returns the current character shear attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*hx,*hy Variables in which the character shear attribute are returned.

Returns:

0 Error
1 OK

Principal errors:

-

GpiSetCharDirection

```
BOOL GpiSetCharDirection (hgpi, direction)
HPS hgpi;
LONG direction;
```

Sets the current character-direction attribute to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

direction

Valid values for *direction* are:

```
CHARDIRECTION_DEFAULT       (0) - Use default
CHARDIRECTION_LEFTTORIGHT   (1) - Left to right
CHARDIRECTION_TOPTOBOTTOM   (2) - Top to bottom
CHARDIRECTION_RIGHTTOLEFT   (3) - Right to left
CHARDIRECTION_BOTTOMTOTOP   (4) - Bottom to top
```

If the specified direction is not valid, the default direction is used.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_CHAR_DIRECTION_ATTR

GpiQueryCharDirection

```
LONG GpiQueryCharDirection (hgpi)
HPS hgpi;
```

Returns the current character direction attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Character direction

Principal errors:

-

GpiSetCharMode

```
BOOL GpiSetCharMode (hgpi, mode)
HPS hgpi;
LONG mode;
```

Sets the current current character-mode attribute to the specified value.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

mode   Valid values for *mode* are:-

```
CHARMODE_DEFAULT   (O)  - Use default
CHARMODE_LOW       (1)  - )
CHARMODE_MED       (2)  - )- see below
CHARMODE_HIGH      (3)  - )
```

CHARPREC_ DEFAULT
        Use the default.

CHARPREC_ LOW
        Use an image character set or font, as determined by the character set attribute. The positioning of characters after the first one is influenced only by the character direction attribute; other character attributes are ignored.

CHARPREC_ MED
        Use an image character set or font, as determined by the character set attribute. The values of character box, character angle, character direction, character shear, character spacing, character extra, character break extra and text alignment are taken into

consideration for the purposes of positioning successive characters, although the individual character definitions are not scaled or rotated.

CHARPREC_ HIGH
Use a vector character set or font, as determined by the character set attribute. All character attributes are followed exactly, both for positioning individual characters and also for scaling, rotating, and shearing them.

If the specified mode is not valid, the default is used.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_CHAR_MODE_ATTR

GpiQueryCharMode

```
LONG GpiQueryCharMode (hgpi)
HPS hgpi;
```

Returns the current character mode attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Character mode

Principal errors:

-

GpiSetCharSpacing

```
BOOL GpiSetCharSpacing (hgpi, width_mult, height_mult)
HPS hgpi;
LONG width_mult;
LONG height_mult;
```

Sets the amount of space or overlap to be provided between successive characters in a string.

The multipliers apply to the character box dimensions (see GpiSetCharBox). They produce increments to the width and height which are applied equally to all characters in the string,

irrespective of any proportional spacing or kerning which may take place.

Only one of the two multipliers will be relevant for any particular character string primitive; this depends upon the character direction (see GpiSetCharDirection). The width increment is used for left-to-right and right-to-left character directions, and the height increment for bottom-to-top and top-to-bottom character directions.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

width_ mult,height_ mult
           The width and height multipliers, respectively. These are each in the form of 4-byte signed fixed point numbers with the high-order word as the integer portion and the low-order word as the fractional portion. Thus, a value of 65536 specifies a multiplier of 1.0.

           The values may be negative, zero, or positive:-

           ● A negative value forces the characters closer together

           ● A value of zero (the default) results in standard spacing

           ● A positive value allows extra space between character boxes

Returns:

   0 Error
   1 OK

Principal errors:

GPIERR_INVALID_CHAR_SPACING_ATTR

GpiQueryCharSpacing

```
BOOL GpiQueryCharSpacing (hgpi, width_mult, height_mult)
HPS hgpi;
LONG *width_mult;
LONG *height_mult;
```

Returns the current character spacing attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*width_mult
        A variable in which the width multiplier is returned.

*height_mult
        A variable in which the height multiplier is returned.

Returns:

0 Error
1 OK

Principal errors:

-

GpiSetCharExtra

```
BOOL GpiSetCharExtra (hgpi, width_extra, height_extra)
HPS hgpi;
LONG width_extra;
LONG height_extra;
```

Sets the amount of space or overlap to be provided between successive characters in a string. The effect of this is identical to the spacing produced by GpiSetCharSpacing, except that the values are additive deltas rather than multipliers. If both character spacing and character extra is used, the effects will be cumulative. Character extra is particularly useful for fonts, where the character box attribute may not be applicable, and are increments to the width and height which are applied equally to all characters in the string, irrespective of any proportional spacing or kerning which may take place.

Only one of the two values will be relevant for any particular character string primitive; this depends upon the character direction (see GpiSetCharDirection). The width increment is used for left-to-right and right-to-left character directions, and the height increment for bottom-to-top and top-to-bottom character directions.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

width_ extra,height_ extra
> The width and height increments, respectively. These
> are each in the form of 4-byte signed fixed point
> numbers with the high-order word as the integer por-
> tion and the low-order word as the fractional portion.
> Thus, a value of 65536 specifies an increment of 1.0.

> The values may be negative, zero, or positive:-

> - A negative value forces the characters closer
>   together

> - A value of zero (the default) results in standard
>   spacing

> - A positive value allows extra space between char-
>   acter boxes

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_CHAR_EXTRA_ATTR
```

GpiQueryCharExtra

```
BOOL GpiQueryCharExtra (hgpi, width_extra, height_extra)
HPS hgpi;
LONG *width_extra;
LONG *height_extra;
```

Returns the current character extra attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*width_ extra
> A variable in which the width increment is returned.

*height_ extra
> A variable in which the height increment is returned.

Returns:

```
0 Error
1 OK
```

Principal errors:

-

GpiSetCharBreakExtra

```
BOOL GpiSetCharBreakExtra (hgpi, code_point,
        width_extra, height_extra)
HPS hgpi;
LONG code_point;
LONG width_extra;
LONG height_extra;
```

Sets the amount of additional space to be provided at a break (normally space) character within a string. The effect of this is additional to any spacing produced by GpiSetCharSpacing and GpiSetCharExtra. The values are additive deltas in world coordinates, and are irrespective of any proportional spacing or kerning which may take place.

Only one of the two values will be relevant for any particular character string primitive; this depends upon the character direction (see GpiSetCharDirection). The width increment is used for left-to-right and right-to-left character directions, and the height increment for bottom-to-top and top-to-bottom character directions.

Character break extra is useful for achieving right justification of a character string.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

code_ point
        The code point of the break character.

width_ extra,height_ extra
        The width and height increments, respectively. These are each in the form of 4-byte signed fixed point numbers with the high-order word as the integer portion and the low-order word as the fractional portion. Thus, a value of 65536 specifies an increment of 1.0.

        The values may be negative, zero, or positive:-

        • A negative value reduces the size of a break character

        • A value of zero (the default) gives the normal size (subject to any other spacing in force)

- A positive value increases the size of a break character

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_CHAR_BREAK_EXTRA_ATTR

GpiQueryCharBreakExtra

```
BOOL GpiQueryCharBreakExtra (hgpi, code_point, width_extra, he.
HPS hgpi;
LONG *code_point;
LONG *width_extra;
LONG *height_extra;
```

Returns the current character break extra attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*code_ point
          A variable in which the break character code point is
          returned.

*width_ extra
          A variable in which the width increment is returned.

*height_ extra
          A variable in which the height increment is returned.

Returns:

0 Error
1 OK

Principal errors:

-

GpiSetTextAlignment

```
BOOL GpiSetTextAlignment (hgpi, horiz, vert)
HPS hgpi;
LONG horiz;
LONG vert;
```

Sets the current alignment, in horizontal and vertical directions,
of subsequently output character strings. The parameters
specify the alignment of character strings horizontally and verti-
cally. Together they define a reference point within the string
that is positioned on the starting point specified for the string.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

horiz   The horizontal alignment. Possible values are

CHARHALIGN_ STANDARD (-1)
        Standard. The alignment depends on the
        current character direction:-

```
Left to right  (0,1)    Left edge of first characte
Top to bottom  (2)      Left edge of first characte
Right to left  (3)      Right edge of first charact
Bottom to top  (4)      Left edge of first characte
```

CHARHALIGN_ DEFAULT (0)
        The default (-1)

CHARHALIGN_ NORMAL (1)
        Normal. The alignment depends on the
        current character direction:-

```
Left to right  (0,1)    Left
Top to bottom  (2)      Center
Right to left  (3)      Right
Bottom to top  (4)      Center
```

CHARHALIGN_ LEFT (2)
        Left alignment. The string is aligned on the
        left edge of its leftmost character.

CHARHALIGN_ CENTER (3)
        Center alignment. The string is aligned on
        the arithmetic mean of left and right.

CHARHALIGN_ RIGHT (4)
        Right alignment. The string is aligned on
        the right edge of its rightmost character.

vert   The vertical alignment. Possible values are

CHARVALIGN_ STANDARD (-1)
        Standard. The alignment depends on the
        current character direction:-

```
Left to right  (0,1)    Bottom edge of first charac
Top to bottom  (2)      Top edge of first character
Right to left  (3)      Bottom edge of first charac
Bottom to top  (4)      Bottom edge of first charac
```

CHARVALIGN_ DEFAULT (0)
> The default (-1)

CHARVALIGN_ NORMAL (1)
> Normal. The alignment depends on the current character direction:-

Left to right	(0,1)	Base
Top to bottom	(2)	Top
Right to left	(3)	Base
Bottom to top	(4)	Base

CHARVALIGN_ TOP (2)
> Top alignment. The string is aligned on the top edge of its topmost character.

CHARVALIGN_ CAP (3)
> Cap alignment. The string is aligned on the cap of its topmost character. Where cap is not defined by the symbol set or font, this is the same as top.

CHARVALIGN_ HALF (4)
> Half alignment. The string is aligned on the arithmetic mean of base and cap.

CHARVALIGN_ BASE (5)
> Base alignment. The string is aligned on the base of its bottom character. Where base is not defined by the symbol set or font, this is the same as bottom.

CHARVALIGN_ BOTTOM (6)
> Bottom alignment. The string is aligned on the bottom edge of its bottom character.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_TEXT_ALIGN_ATTR

The terms "top left", "bottom right", and so on, are well defined when the character angle and the direction of the co-ordinate system are such that the baseline is parallel to the x axis, running from left ot right on the device, and there is no character shear.

If the character is rotated or sheared, the term "top left" applies to the corner of the character box that appears in the top left when no rotation or shear is applied.

GpiQueryTextAlignment

```
BOOL GpiQueryTextAlignment (hgpi, horiz, vert)
HPS hgpi;
LONG *horiz;
LONG *vert;
```

Returns the current horizontal and vertical text alignment values.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi	The handle for the GPI presentation space.
*horiz	A variable in which the horizontal text alignment is returned.
*vert	A variable in which the vertical text alignment is returned.

Returns:

0 Error
1 OK

Principal errors:

-

7.1.21.5 Primitive Functions

GpiCharString

```
SHORT GpiCharString (hgpi, n, ch)
HPS hgpi;
LONG n;
LPSTR ch;
```

GpiCharStringAt

```
SHORT GpiCharStringAt (hgpi, x0, y0, n, ch)
HPS hgpi;
LONG x0;
LONG y0;
LONG n;
LPSTR ch;
```

Draws a character string starting at the current x,y position (GpiCharString), or at the specified position (GpiCharStringAt).

Current position is updated to the point at which the next character would have been drawn, had there been one.

This function can occur within the picture; for example if it is

issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
x0,y0	The starting position (GpiCharStringAt only).
n	The number of characters in the string.
ch	The string of character codepoints.

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

```
GPIERR_DBCS_SYMBOL_SET_NOT_AVAILABLE (warning)
GPIERR_INVALID_CHAR_STRING_LENGTH
GPIERR_INVALID_DBCS_CHAR_IN_STRING (warning)
GPIERR_DBCS_CHAR_STRING_MUST_HAVE_EVEN_LENGTH (warning)
```

GpiCharStringPos

```
SHORT GpiCharStringPos (hgpi, rect, options,
        n, ch, dx)
HPS hgpi;
GRECT rect;
ULONG options;
LONG n;
LPSTR ch;
LONG dx[];
```

GpiCharStringPosAt

```
SHORT GpiCharStringPosAt (hgpi, x0, y0, rect, options, n, ch,
HPS hgpi;
LONG x0;
LONG y0;
GRECT rect;
ULONG options;
LONG n;
LPSTR ch;
LONG dx[];
```

Draws a character string starting at the current x,y position (GpiCharStringPos), or at the specified position (GpiCharString-PosAt).

A vector of increments may optionally be specified. which allows control over the positioning of each character after the first. These are distances measured in world co-ordinates (along the baseline for left-to-right and right-to-left character directions, and along the shearline for top-to-bottom and bottom-to-top).

The i'th increment is the distance of the reference point (eg bottom left corner) of the (i+1)'th character from the reference point of the i'th. The last increment may be needed to update current position.

These increments, if specified, set the widths of each character. Any spacing called for by

- GpiSetCharSpacing

- GpiSetCharExtra

- GpiSetCharBreakExtra

is applied in addition to the widths defined by the vector.

A further option allows a rectangle to be specified, which is to be used as the background of the string, rather than using the normal method of defining the background. This rectangle will be painted using the current character background color and an overpaint mix (unless this is in a dynamic segment, when leave-alone will be used). Both corners of the rectangle are specified, so that the rectangle is positioned independently of current position.

A further option allows clipping of the string to the rectangle. This is independent of whether the rectangle is actually drawn.

The string may optionally be drawn de-emphasized. This is used in menus to denote options which are not currently selectable. The implementation may choose whichever method of de-emphasis is most appropriate on the particular device.

Current position may optionally be updated to the point at which the next character would have been drawn, had there been one, or it can be left unchanged by this function.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
x0,y0	The starting position (GpiCharStringPosAt only).
rect	A rectangle structure defining the two corners of the rectangle, which defines the background of the characters. Ignored if CHS_OPAQUE is B'0' and CHS_CLIP is B'0'.
options	Formatting options. This consists of 32 flags (with 0 the least significant). These may be used in combination. Each set bit has the following meaning:-

CHS_ OPAQUE (bit 0)

> Background of characters is defined by rectangle with corners at *(x0,y0)*, *(x1,y1)*, rather than by the characters as usual. The rectangle is to be shaded (with background color and overpaint) before drawing.

CHS_ VECTOR (bit 1)

> Increments vector supplied (*dx*). If this bit is zero, *dx* is ignored

CHS_ DEEMPHASIZED (bit 2)

> String to be drawn de-emphasized. This is used for selection items to indicate that the item is not currently selectable.

CHS_ LEAVEPOS (bit 3)

> Leave current position at the start of the string. If this bit is not set, current position is moved to the position at which the next character would have been drawn, if there had been one.

CHS_ CLIP (bit 4)

> Clips the string to the rectangle if set.

Other bits are reserved and must be zero.

n	The number of characters in the string.
ch	The string of character codepoints
dx[n]	A vector of (n values of) increment values. These are 4-byte signed integers, in world co-ordinates.

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

```
GPIERR_DBCS_SYMBOL_SET_NOT_AVAILABLE (warning)
GPIERR_INVALID_CHAR_STRING_LENGTH
GPIERR_INVALID_DBCS_CHAR_IN_STRING (warning)
GPIERR_DBCS_CHAR_STRING_MUST_HAVE_EVEN_LENGTH (warning)
GPIERR_INVALID_POSITIONING_VALUE
```

7.1.22 Marker Functions

Functions described in this section are for drawing markers, and for controlling the attributes with which they are drawn.

Markers are drawn with the following attributes:-

- Marker color
- Marker background color
- Marker mix
- Marker background mix
- Marker set
- Marker symbol
- Marker box

The default marker attributes are:-

```
Marker color:                Color 7
Marker background color:      Color 0
Marker mix:                   Overpaint
Marker background mix:        Leave alone
Marker set:                   Standard 11-value set
Marker symbol:                Cross
Marker box:                   Device dependent
```

Markers may be drawn using either *symbol sets* or *fonts*. For further information see the section, "Character Functions".

7.1.22.1 Attribute Setting Functions

GpiSetMarkerSet

```
BOOL GpiSetMarkerSet (hgpi, set)
HPS hgpi;
LONG set;
```

Sets the current marker-set attribute to the specified value.

These functions can occur within the picture; for example if either is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi The handle for the GPI presentation space.

set The identity (lcid) of the required marker set:

> *0* Default set.
>
> *64* Range of values for
> *thru* the ID of a loaded
> *239* symbol set.
>
> *240* Base marker set.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MARKER_SET_ATTR

GpiQueryMarkerSet

```
LONG GpiQueryMarkerSet (hgpi)
HPS hgpi;
```

Returns the current marker set attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi The handle for the GPI presentation space.

Returns:

-1 Error
>=0 Marker set

Principal errors:

-

GpiSetMarker

```
BOOL GpiSetMarker (hgpi, symbol)
HPS hgpi;
LONG symbol;
```

Sets the current value of the marker-symbol attribute to the specified value.

These functions can occur within the picture; for example if either is issued in store or draw-and-store mode,

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

symbol   The identity of the required marker symbol. A value
          of *0* selects the default marker symbol, any other value
          identifies a symbol in the current marker set. Valid
          values in the base marker set are:

```
MARKSYM_CROSS          ( 1) - Cross.
MARKSYM_PLUS           ( 2) - Plus.
MARKSYM_DIAMOND        ( 3) - Diamond.
MARKSYM_SQUARE         ( 4) - Square.
MARKSYM_SIXPOINTSTAR   ( 5) - Six-point star.
MARKSYM_EIGHTPOINTSTAR ( 6) - Eight-point star.
MARKSYM_SOLIDDIAMOND   ( 7) - Filled diamond.
MARKSYM_SOLIDSQUARE    ( 8) - Filled square.
MARKSYM_DOT            ( 9) - Dot.
MARKSYM_SMALLCIRCLE    (10) - Small circle.
MARKSYM_BLANK          (64) - Blank.
```

Returns:

> *0* Error
> *1* OK

Principal errors:

GPIERR_INVALID_MARKER_ATTR

GpiQueryMarker

```
LONG GpiQueryMarker (hgpi)
HPS hgpi;
```

Returns the current marker-symbol attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

Returns:

> *-1* Error
> *>=0* Marker symbol

Principal errors:

-

GpiSetMarkerBox

```
BOOL GpiSetMarkerBox (hgpi, width, height)
HPS hgpi;
LONG width;
LONG height;
```

Sets the current marker-box attribute to the specified value.

These functions can occur within the picture; for example if

either is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

The attribute mode (see GpiSetAttrMode) determines whether or not the push form of the function is generated.

Parameters:

hgpi   The handle for the GPI presentation space.

width,height
        The marker box dimensions in world co-ordinates. These are long signed integers, with a notional binary point between the second and third bytes. Thus a width of 8 world co-ordinate units is represented as 8*65536.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_MARKER_BOX_ATTR

GpiQueryMarkerBox

```
BOOL GpiQueryMarkerBox (hgpi, width, height)
HPS hgpi;
LONG *width;
LONG *height;
```

Returns the current marker-box attribute.

This function is invalid in store mode or implicit draw mode.

Parameters:

hgpi   The handle for the GPI presentation space.

*width,*height
        Variables in which the marker box dimensions are returned.

Returns:

0 Error
1 OK

Principal errors:

-

7.1.22.2   Primitive Functions

GpiMarker

```
SHORT GpiMarker (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

Draws a marker, with its center at the specified position.

Current position is moved to the specified point.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi   The handle for the GPI presentation space.

x,y   The co-ordinates of the point at which the marker is to be drawn.

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

-

GpiPolyMarker

```
SHORT GpiPolyMarker (hgpi, n, x, y)
HPS hgpi;
LONG n;
LONG x[];
LONG y[];
```

Draws a series of one or more markers, at each of the specified positions.

The center of the marker is drawn at the specified position(s).

Upon completion, the current x,y position is the x,y position of the last marker in the series.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi The handle for the GPI presentation space.

n The number of x,y pairs.

x An array of integer values (x coordinates).

y An array of integer values (y coordinates).

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

```
GPIERR_INVALID_ARRAY_COUNT
```

7.1.23 Image Functions

Functions described in this section are for drawing images, and for controlling the attributes with which they are drawn.

Image attributes are also used to control GpiBitBlt and GpiStretchBlt functions (see the section, "Bitmap Support"), and also GpiPaintRgn (see the section, "Region Support").

Images are drawn with the following attributes:-

- Image color
- Image background color
- Image mix
- Image background mix

The manipulation of the color, background color, mix, and background mix attributes globally is described in the section, "Color and Mix Functions".

The default image attributes are:-

```
Image color:              Color 7
Image background color:   Color 0
Image mix:                Overpaint
Image background mix:     Leave alone
```

7.1.23.1   Primitive Functions

GpiImage

```
SHORT GpiImage (hgpi, format, width, depth, length, data)
HANDLE hgpi;
LONG format;
LONG width;
LONG depth;
LONG length;
LPBUF data;
```

Draws a rectangular image with the top left corner at current position.

The width and height specify the size of the image in pixels. The data contains pixel data for each of the *depth* rows, starting with the top row. For each row, the data contains one bit per pixel, left to right, with *width/8* bytes, rounded up if the width is not a multiple of 8. If, for example, the image width specified is 12, each row of data must be padded out to a length of 16 so that the data in the row occupies 2 bytes exactly.

The current x,y position is not changed.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, order(s) will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
format	Format of image data. This is a reserved field that should be specified as zero.
width	Width of image area (specified as a number of pels).
depth	Depth of image area (specified as a number of pels).
length	Length of data, in bytes
data	Image data as described above.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_IMAGE_FORMAT
GPIERR_INVALID_IMAGE_DIMENSION
GPIERR_INVALID_IMAGE_DATA_LENGTH
```

7.1.24   Miscellaneous Functions

GpiComment

```
BOOL GpiComment (hgpi, n, data)
HPS hgpi;
LONG n;
LPBUF data;
```

Indicates a Comment drawing order containing the specified data.

This function can occur within the picture; for example if it is issued in store or draw-and-stored mode, an order will be constructed and placed into the current segment.

Parameters:

hgpi	The handle for the GPI presentation space.
n	The length of *data* in bytes. *n* must not be greater than 255.
data	A pointer to the comment string

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_COMMENT_LENGTH

GpiSegmentCharacteristics

```
BOOL GpiSegmentCharacteristics (hgpi, chid, length, data)
HPS hgpi;
LONG chid;
LONG length;
LPBUF data;
```

This order provides the facility to specify architected or user defined characteristics for a segment.

This function can occur within the picture; for example if it is issued in store or draw-and-store mode, an order will be constructed and placed into the prologue of the current segment.

Parameters:

hgpi   The handle for the GPI presentation space.

chid   The identification code for the characteristics (X'00' - X'7F' for architected characteristics and X'80' - X'FF' for user-defined characteristics).

length   The length of data in *data*

data   A pointer to a buffer containing the characteristics data.

Returns:

0 Error
1 OK

Principal errors:

`GPIERR_INVALID_SEG_CHARACTERISTICS_LENGTH`

GpiQueryDDALength

```
LONG GpiQueryDDALength (hgpi, type)
HPS hgpi;
LONG type;
```

This returns the length of the storage data block which the system needs to hold the intermediate values for a particular DDA sequence.

A DDA sequence should consist of the following:-

1. Issue GpiQueryDDALength to determine the system block length required

2. Acquire storage for the system block of this length

3. Initialise the block -

    *   First 4 bytes are zero

    *   Parameters for the particular DDA request (eg line endpoints)

4. Issue GpiDDA as many times as necessary, until the end of the sequence is reached. The system block should not be altered while this sequence is in progress.

Parameters:

hgpi   The handle for the GPI presentation space.

type   Specifies which DDA sequence is going to be issued, as follows:-

1 GpiLineDDA
2 GpiFilletDDA
3 GpiPartialArcDDA
4 GpiArcDDA
5 GpiFilletSharp

6 GpiSpline

Returns:

-1 Error
>=0 Length required for the system block

Principal errors:

GPIERR_INVALID_DDA_TYPE

GpiDDA

```
LONG GpiDDA (hgpi, type, sys_block, n, points)
HPS hgpi;
LONG type;
LPBUF sys_block;
LONG n;
LPBUF points;
```

This causes a DDA request to be performed, within a sequence (see GpiQueryDDALength).

This is a request for the system to trace the outline of a figure. Instead of drawing it, the system passes the co-ordinates of the points which would have been drawn, back to the application.

Attributes are not taken into account for a DDA sequence. All current transformations, however (including arc parameters if applicable) *are* taken into consideration. The input co-ordinates are in World Co-ordinates, the output co-ordinates are in device co-ordinates.

A DDA sequence should consist of the following:-

- Issue GpiQueryDDALength to determine the system block length required

- Acquire storage for the system block of this length

- Initialise the system block -

  - First 4 bytes are zero

  - Parameters for the particular DDA request (eg line endpoints). See below for details.

- Issue GpiDDA as many times as necessary, until the number of points returned is less than the number for which the application has allowed. The system block should not be altered while this sequence is in progress.

Parameters:

hgpi   The handle for the GPI presentation space.

type   Specifies which type of DDA is required, as follows:-

    1 GpiLineDDA

2 GpiFilletDDA
3 GpiPartialArcDDA
4 GpiArcDDA
5 GpiFilletSharp
6 GpiSpline

Unpredictable results will occur if this is changed within a particular sequence.

sys_ block

The address of the system block, which must be of adequate length (as returned by GpiQueryD-DALength).

The start of *sys_ block* must be initialised before the first GpiDDA call in the sequence, and not changed again before the end of the sequence. The values depend upon the DDA type, as follows:-

- For *Line DDA* :-

  - Bytes 00-03: Zeroes

  - Bytes 04-07: x co-ordinate of start point

  - Bytes 08-0B: y co-ordinate of start point

  - Bytes 0C-0F: x co-ordinate of end point

  - Bytes 10-13: y co-ordinate of end point

- For *Fillet DDA* :-

  - Bytes 00-03: Zeroes

  - Bytes 04-07: x co-ordinate of start point

  - Bytes 08-0B: y co-ordinate of start point

  - Bytes 0C-0F: x co-ordinate of join point

  - Bytes 10-13: y co-ordinate of join point

  - Bytes 14-17: x co-ordinate of end point

  - Bytes 18-1B: y co-ordinate of end point

  See GpiPolyFillet for more details.

- For *Partial Arc DDA* :-

  - Bytes 00-03: Zeroes

  - Bytes 04-07: x co-ordinate of line start

  - Bytes 08-0B: y co-ordinate of line start

  - Bytes 0C-0F: x co-ordinate of centre

  - Bytes 10-13: y co-ordinate of centre

- Bytes 14-17: multiplier
- Bytes 18-1B: start angle
- Bytes 1C-1F: end angle

See GpiPartialArc for more details.

- For *Arc DDA* :-
  - Bytes 00-03: Zeroes
  - Bytes 04-07: x co-ordinate of start point
  - Bytes 08-0B: y co-ordinate of start point
  - Bytes 0C-0F: x co-ordinate of intermediate point
  - Bytes 10-13: y co-ordinate of intermediate point
  - Bytes 14-17: x co-ordinate of end point
  - Bytes 18-1B: y co-ordinate of end point

See GpiArc for more details.

- For *FilletSharp DDA* :-
  - Bytes 00-03: Zeroes
  - Bytes 04-07: x co-ordinate of start point
  - Bytes 08-0B: y co-ordinate of start point
  - Bytes 0C-0F: x co-ordinate of join point
  - Bytes 10-13: y co-ordinate of join point
  - Bytes 14-17: sharpness
  - Bytes 18-1B: x co-ordinate of end point
  - Bytes 1C-1F: y co-ordinate of end point

See GpiPolyFilletSharp for more details.

- For *Spline DDA* :-
  - Bytes 00-03: Zeroes
  - Bytes 04-07: x co-ordinate of start point
  - Bytes 08-0B: y co-ordinate of start point
  - Bytes 0C-0F: x co-ordinate of 1st control point
  - Bytes 10-13: y co-ordinate of 1st control point
  - Bytes 14-17: x co-ordinate of 2nd control point

- Bytes 18-1B: y co-ordinate of 2nd control point
- Bytes 1C-1F: x co-ordinate of end point
- Bytes 20-23: y co-ordinate of end point

See GpiPolySpline for more details.

n   The maximum number of co-ordinate pairs for which there is room in *points*.

points   The address of a buffer in which the (x,y) co-ordinates of the next *n* points in the sequence will be returned, as long integers in local format.

Returns:

```
-1 Error
>=0 Number of points returned (this may be less thann,
    if the end of the sequence has been reached. The applicati
    should check this to detect the end of the sequence.
```

Principal errors:

```
GPIERR_INVALID_DDA_TYPE
GPIERR_INVALID_ARRAY_COUNT
```

7.1.25 Bitmap Support

Bitmaps can either be created in memory, or loaded from file. At this stage the only operations which may be performed on the bitmap are to get and set the contents of the bitmap, as a stream of bytes.

A bitmap can be selected into a Device Context, and the Device Context associated with a presentation space (GpiAssociate or VioAssociate). Copy (GpiBitBlt) operations may now be performed, with the bitmap as a source or target (or both). The presentation space can also be drawn into the bitmap.

Certain devices, notably a raster screen, may be treated as bitmaps, and the same BitBlt operations performed on them, through their associated presentation spaces.

7.1.25.1 Bitmap Operations

(The term 'raster operations' is also used elsewhere in this document to refer to bitmap operations.)

- A bitmap can be created and destroyed using the GpiCreateBitmap and GpiDestroyBitmap functions.
- A bitmap can be loaded from a resource using the GpiLoadBitmap function.

- A bitmap may be selected into a memory Device Context using the GpiSelectBitmap function.

- A GPI presentation space may be drawn into a Device Context associated with a bitmap using the GpiDraw.. functions

- An Advanced Vio presentation space may be drawn into a Device Context associated with a bitmap using the VioShowPS or VioShowBuf functions

- A source bitmap, selected into a suitable Device Context, and associated with either a Gpi or Vio presentation space, may be copied into another suitable Device Context (also associated with either a Gpi or Vio presentation space) using the GpiBitBlt function

- A bitmap may be copied to/from application storage using the Gpi-GetBitmapBits and GpiSetBitmapBits functions. The format of data in application storage is the same as the format of the data created by the bitmap editor. Thus the application may read the bitmap defintion created by the bitmap editor, and pass the data to the bitmap interface.

- Operations are provided to cause an individual pel to be set or retrieved, and also to perform a flood fill operation.

As an example, to create a bitmap, draw into it, and then copy this drawing to a window on the screen, the following must be done:-

1. DevOpenDC

   Create a memory DC, compatible with the screen

2. GpiCreateBitmap

   Create a bitmap, in a format also compatible with the screen

3. GpiSelectBitmap

   Select the bitmap into the memory DC

4. GpiCreatePS

   Create a Gpi presentation space

5. GpiAssociate

   Associate this presentation space with the memory DC

6. Draw functions to the Gpi presentation space

   The bitmap receives the rastered image of the drawing operations

7. (Assume that a presentation space is already available for drawing to the screen window)

8. GpiBitBlt

   Copy from one bitmap (presentation space) to the screen window

7.1.25.2 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 for-
mats commonly used by raster devices. Only single plane formats are
standard, but it is very easy to convert these to any multiple plane format
used internally by a device.

The pixel data is stored in the bitmap in the order of the coordinates as
they would appear on a display screen. That is, the pixel in the lower left
corner is the first in the bitmap. Pixels are scanned to the right and up
from there. The first pixel's bits are stored beginning in the most
significant bits of the first byte. The data for pixels in each scan line is
packed together tightly. Each scanline, however, will be padded at the
end so that each scan line begins on a ULONG boundary.

7.1.25.3 Bitmap Info 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 in pixels
   UINT   BitmapHeight;     /* number of scanlines (pixels)
   UINT   BitmapPlanes;     /* number of planes (1 if standard format)
   UINT   BitmapBitcount;   /* number of adjacent bits per pixel
   RGB    BitmapColors[];   /* color table
}BMINFO;
```

The BitmapColors array is a packed array of 24 bit RGB values. If there
are N bits per pixel, then the BitmapColors array will 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.

Some calls use a structure which is like BMINFO but does not have the
BitmapColors array. This is defined as follows:-

```
struct BitmapInfoHeader {
```

```
    UINT   BitmapWidth;        /* length of a scanline in pixels    */
    UINT   BitmapHeight;       /* number of scanlines (pixels)      */
    UINT   BitmapPlanes;       /* number of planes (1 if standard format) */
    UINT   BitmapBitcount;     /* number of adjacent bits per pixel */
}BMINFOH;
```

7.1.25.4 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
```

```
ULONG ExampleBitmap[] = {
    0x23,0x12,0x30,0x00              /* bottom line */
    0x31,0x23,0x10,0x00              /* middle line */
    0x12,0x31,0x20,0x00              /* top line    */
};

#define BLACK    0x000000L
#define RED      0x0000FFL
#define GREEN    0x00FF00L
#define BLUE     0xFF0000L

struct BitmapInfoTable ExampleInfo = {
        5,                          /* width */
        3,                          /* height */
        1,                          /* planes */
        4,                          /* bitcount */
        BLACK,RED,GREEN,BLUE,       /* color table */
        BLACK,BLACK,BLACK,BLACK,
        BLACK,BLACK,BLACK,BLACK,
        BLACK,BLACK,BLACK,BLACK
};
```

7.1.25.5 Uses for bitmaps

Uses of Bitmaps include:

- Menus which can be put on screen and removed very rapidly.

- Generation of images from graphical data for subsequent storage and
  transmission.

- Creation and use of application symbols.

- Rapid re-healing of a picture when an overlaying Window is removed.

- Animation.

Although the operations described in this section will be recorded in a metafile, the results of them are likely to be device dependent, and may cause significantly different results between devices with different resolutions as well as color capabilities.

7.1.25.6 Creation and Selection Funtions

GpiLoadBitmap

```
HBITMAP GpiLoadBitmap (hdc, idModule, bitmapid, width, height)
HDC hdc;
UINT idModule;
UINT bitmapid;
LONG width;
LONG height;
```

This loads a bitmap from a resource, and returns the bitmap handle. The bitmap is now available for selection into a Device Context.

Optionally, the bitmap will be stretched to the specified size.

The bitmap may have been created by the bitmap editor.

Parameters:

---

hdc   The handle of a DC which identifies a device whose memory should be used, if possible, to hold the bitmap. If it is not possible, main memory will be used, but the bitmap will be in a format compatible with the specified device.

idModule
       The module handle of the dynlink resource library. If this is NULL, the application resource file is used.

bitmapid
       The id of the bitmap within the resource file.

width,height
       These integers define the width and height of the bitmap in pels. If either is zero, the bitmap will retain its original size. Otherwise, it will be stretched to the specified *width* and *height*.

Returns:

```
 0 Error
!=0 Bitmap handle
```

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_RESOURCE_HANDLE
```

GPIERR_INVALID_RESOURCE_ID

GpiCreateBitmap

```
HBITMAP GpiCreateBitmap (hdc, info_hdr, usage)
HDC hdc;
BMINFOH info_hdr;
ULONG usage;
```

This creates a Bitmap of the specified form and returns the bit-map handle.

On some devices it may be possible to create the bitmap in the device's own memory. A non-null DC handle may be supplied to indicate that this should be done if possible. The DC can be any DC connected to the device in question (eg any window DC for the screen).

There are a number of standard bitmap formats which should normally be adhered to. See the section, "Standard Bitmap Formats". Another format may be used if it is known that the device supports it.

The application may also indicate whether the system is at liberty to discard this bitmap in the event of a memory shortage.

The bitmap is now available for selection into a Device Context.

Parameters:

hdc   The handle of a DC which identifies a device whose memory should be used, if possible, to hold the bit-map. If it is not possible, main memory will be used, but the bitmap will be in a format compatible with the specified device.

info_hdr
         A Bitmap Info Header structure which defines the for-mat of the bitmap to be created. It contains the width, height, number of planes, and bitcount:

BitmapWidth, BitmapHeight
         These integers define the width and height of the Bitmap in pels.

BitmapPlanes
         The number of color planes in the bitmap. Each plane logically contains (width*height*bitcount) bits (the actual length may be greater because of padding)

BitmapBitcount
         The number of color bits per pel, within one plane.

usage   Usage information for this bitmap as follows:-

BM_ BACKUP (Bit 0)
        Memory backup

0   No need to retain this bitmap in PC storage while it is in device storage

1   Keep memory backup while in device storage

BM_ DISCARD (Bit 1)
        Discardability

0   May discard this bitmap if short of storage (providing the bitmap is not either selected into a Device Context, or in use as the currently selected area fill pattern. Note that the bitmap *can* be discarded if it has an lcid associated with it, but is not the currently selected fill pattern. In general, if a bitmap is to be used for area filling, it should not be designated as discardable.)

1   May not discard this bitmap

If the bitmap is stored on a device, the *usage* parameter is passed to the device. Bits 16-31 may be used for special reasons known to be supported by the particular device driver.

Returns:

0 Error
!=0 Bitmap handle

Principal errors:

GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_BITMAP_PARAMETER
GPIERR_INVALID_USAGE

GpiDeleteBitmap

```
BOOL GpiDeleteBitmap (hbm)
HBITMAP hbm;
```

This destroys the specified bitmap.

Parameters:

hbm   The handle of the bitmap to be deleted.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_BITMAP_HANDLE
GPIERR_CANNOT_DELETE_SELECTED_BITMAP

GpiSelectBitmap

HBITMAP GpiSelectBitmap (hdc, hbm)
HDC hdc;
HBITMAP hbm;

This selects the specified bitmap into the specified memory Device Context.

The Device Context may represent a different physical device from the one which the bitmap was originally loaded or created on, providing its format is convertible to one supported on the new device. This is guaranteed if one of the standard formats has been used.

Following this function, the bitmap must be associated (using GpiAssociate or VioAssociate) before it can be used for drawing into or copying to/from.

If there is already a bitmap selected into the Device Context, then the handle of this bitmap will be returned, before the new bitmap is selected in.

It is an error if the new bitmap is already selected into any Device Context.

Parameters:

hdc   The handle of the Device Context.

hbm   The handle of the bitmap to be selected. A null handle causes the bitmap currently selected into this DC to become deselected.

Returns:

-1 Error
0 OK (null handle)
<-1 Old bitmap handle
>0 Old bitmap handle

Principal errors:

GPIERR_INVALID_BITMAP_HANDLE
GPIERR_INVALID_DC_HANDLE
GPIERR_BITMAP_AND_DC_INCOMPATIBLE

GpiSetBitmapDimension

```
BOOL GpiSetBitmapDimension (hbm, size)
HBITMAP hbm;
LONG size[];
```

This associates a width and height with a bitmap, in 0.1 mm units. These values are not used internally by the system, but are retained with the bitmap. GpiQueryBitmapDimension can be used to retrieve them.

Parameters:

hbm   The handle of the bitmap.

size[2] A two-element array which contains two unsigned integers, which are the width and height, respectively, of the bitmap in 0.1 mm units.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_BITMAP_HANDLE
```

GpiQueryBitmapDimension

```
BOOL GpiQueryBitmapDimension (hbm, size)
HBITMAP hbm;
LONG size[];
```

This returns the width and height of a bitmap, as specified on a previous GpiSetBitmapDimension.

Parameters:

hbm   The handle of the bitmap.

size[2] A two-element array which on return contains two unsigned integers, the width and height, respectively, of the bitmap in 0.1 mm units.

         If GpiSetBitmapDimension has not been used to set these values, zeroes will be returned.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_BITMAP_HANDLE
```

7.1.25.7   Operations on Raw Bitmaps

GpiQueryDeviceBitmapFormats

```
BOOL GpiQueryDeviceBitmapFormats (hdc, count, data)
HDC hdc;
LONG count;
LONG data[];
```

This returns the formats of bitmaps which may participate in GpiBitBlt operations with the specified device class. This will normally be a smaller set than the standard bitmap formats (see the section, "Standard Bitmap Formats").

The number of device bitmap formats may be found from DevQueryCaps.

The first pair of (planes, bitcount) returned will be the one which most closely matches the device.

Parameters:-

hdc The handle of a Device Context for the class of device for which formats are required. This must either be a memory Device Context, or a Device Context for a device which supports raster operations.

count The number of elements in *data*. It must be an even number. To get the complete set of formats returned, it must be at least double the number of device formats returned by DevQueryCaps.

data[count]
 An array of elements which, on return, is set to pairs of (number of planes, bitcount), for each supported format in turn. Any excess elements are set to zero.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_ARRAY_COUNT
```

GpiQueryBitmapParameters

```
BOOL GpiQueryBitmapParameters (hbm, info_hdr)
HBITMAP hbm;
BMINFOH info_hdr;
```

Returns information about the the bitmap identified by the specified bitmap handle.

Parameters:-

hbm   The handle of the bitmap.

info_hdr

      A Bitmap Info Header structure which on return will
have been filled in with data about the specified bit-
map. The structure is the first four elements (width,
height, planes, bitcount) of a Bitmap Info Table (see
the section, "Bitmap Info Tables").

Returns:

  0 Error
  1 OK

Principal errors:

GPIERR_INVALID_BITMAP_HANDLE

GpiGetBitmapBits

```
LONG GpiGetBitmapBits (hdc, scan_start, scan_count,
          address, info)
HDC hdc;
LONG scan_start;
LONG scan_count;
LPBUF address;
BMINFO info;
```

This transfers bitmap data from the specified Device Context to
application storage. The Device Context must be a memory
Device Context, with a bitmap currently selected.

The Bitmap Info Table (see the section, "Bitmap Info Tables")
must be initialized by the application with the values of
BitmapPlanes and *BitmapBitcount*, for the format of data which
it wants. This must be one of the standard formats (see the sec-
tion, "Standard Bitmap Formats"). On return, *BitmapWidth*,
BitmapHeight, and the *BitmapColors* array will have been filled
in by the system.

Conversion of the bitmap data will have been carried out if
necessary.

address must point to a storage area large enough to contain
data for the requested number of scanlines. The amount of
storage required for one scanline can be determined by calling
GpiQueryBitmapParameters. It is

```
((bitcount*width + 31)/32) *height*planes*4   bytes
```

Parameters:

hdc The handle of the Device Context.

scan_ start
> The scan-line number at which the data transfer is to start.

scan_ count
> The number of scan lines to be returned.

address The address in application storage into which the bit-map data is copied.

info The address in application storage of a Bitmap Info Table as described above.

Returns:

-1 Error
$>=0$ Number of scanlines actually returned

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_SCAN_COUNT
GPIERR_INVALID_START_SCAN
GPIERR_INVALID_INFO_TABLE
GPIERR_INVALID_DC_TYPE
GPIERR_NO_BITMAP_SELECTED_INTO_DC
```

GpiSetBitmapBits

```
LONG GpiSetBitmapBits (hdc, scan_start, scan_count,
      address, info)
HDC hdc;
LONG scan_start;
LONG scan_count;
LPBUF address;
BMINFO info;
```

This transfers bitmap data from application storage into the specified Device Context.

The Device Context must be a memory Device Context, with a bitmap currently selected. Note that this function will not set bits directly to any other kind of device.

If the format of the supplied bitmap does not match that of the device, it is converted, using the supplied Bitmap Info Table. Only the standard formats will be supported (see the section, "Standard Bitmap Formats").

Parameters:

hdc The handle of the Device Context.

scan_ start
> The scan-line number at which the data transfer is to start.

scan‚ count
> The number of scan lines to be transmitted.

address   The address in application storage from which the bit-map data is to be copied.

info   The address in application storage of the Bitmap Info Table (see the section, "Bitmap Info Tables").

Returns:

-1 Error
$>=0$ Number of scanlines actually set

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_SCAN_COUNT
GPIERR_INVALID_START_SCAN
GPIERR_INVALID_INFO_TABLE
GPIERR_INVALID_DC_TYPE
GPIERR_NO_BITMAP_SELECTED_INTO_DC
```

7.1.25.8   Operations through Presentation Spaces

GpiBitBlt

```
SHORT GpiBitBlt (hgpi_targ, hgpi_src, n,
        xy_array, rop, mode)
HPS hgpi_targ;
HPS hgpi_src;
LONG n;
LONG xy_array[];
LONG rop;
LONG mode;
```

This copies a rectangle of bitmap image data from a bitmap selected into a Device Context associated with the source presentation space to a bitmap selected into a Device Context associated with the target presentation space. Alternatively, either presentation space may be associated with a Device Context which specifies a suitable raster device, for example the screen. (It is an error if either device does not support raster operations.)

Unless the device is a banded printer, both source and target may refer to the same presentation space. If so, the copy will be non-destructive if the source and target rectangles overlap.

A rectangle may be specified in both the source and target presentation spaces, in world co-ordinates for the respective presentation spaces. These values are transformed, using all of the appropriate transforms, into device co-ordinates. The resulting rectangles are non-inclusive, so that they include the

left and lower boundaries in device space, 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.

Depending upon the *mode*, stretching and/or compressing of the data may occur. If the *mode* specifies *0*, no stretching or compressing of the data occurs, and the size of the data copied from the source in device units is equal to the size of the target rectangle in device units. The top right of the source rectangle, if specified, is ignored.

For *mode* values of *1 - 3*, if the size of the source rectangle is different from that of the target rectangle (in device units), the source data will be stretched or compressed as necessary in each dimension (the pattern data is never stretched or compressed). *mode* specifies how any eliminated rows/columns of bits are to be treated.

Values of *mode* $>= 32768$ are not necessarily invalid. The parameters are passed on to the device driver, so that advantage may be taken of any special device capability known to be supported on a particular device.

The following current attributes of the target presentation space are used:-

— Pattern color

— Pattern background color

— Pattern set

— Pattern symbol

The color values are used in converting between monochrome and color data. This is the only format conversion performed by GpiBitBlt. The conversions are as follows:-

- Outputing a monochrome pattern to a color device

    In this case the pattern is converted first to a color pattern, using the current pattern colors:

    — source 1s -> pattern foreground color

    — source 0s -> pattern background color

- BltBlting from a monochrome bitmap to a color bitmap (or device)

    The source bits are converted as follows:

    — source 1s -> image foreground color

    — source 0s -> image background color

- BltBlting from a color bitmap to a monochrome bitmap (or device)

— pels which are the source background color -> image background color

— all other pels -> image foreground color

Note that in all of these cases it is the attributes of the *target* presentation space which are used.

If the mix (*rop*) does not call for a pattern, then the pattern set and pattern symbol are not used. If it does not require a source (this is invalid for *mode = 1 - 3*), then *hgpi_src* is not required and must be null. Sx1,Sy1 are also ignored in this case. An example where neither the source nor the pattern is required is when (part of) a bitmap is to be cleared to a particular color.

If the mix does require both source and pattern, then a 3-way operation is performed.

If any of the source data is not available, for example if the source presentation space is connected to a screen window, and the source rectangle is not currently all visible, the contents of the unavailable parts are undefined. GpiRectVisible can be used if necessary, prior to the GpiBitBlt operation, to see if this will be the case.

If this function occurs while the drawing mode in *hgpi_src* is store or draw-and-store, order(s) will be constructed and placed in the current segment. Note, however, that this function is very device-dependent.

Parameters:

hgpi_targ
> The handle of the target Gpi presentation space.

hgpi_src
> The handle of the source Gpi presentation space.

n
> The number of (x,y) points specified in *xy_array*. For *mode = 0*, this must be at least *3*, and for *mode = 1 - 3*, it must be at least *4*.

xy_array[]
> An array of *n* (x,y) points, in the order *Tx1,Ty1,Tx2,Ty2,Sx1,Sy1*. These are as follows:-

> Tx1,Ty1
>> Specify the bottom left corner of the target rectangle in target world co-ordinates.

> Tx2,Ty2
>> Specify the top right corner of the target rectangle in target world co-ordinates.

> Sx1,Sy1   Specify the bottom left corner of the source rectangle in source world co-ordinates.
>
> Sx2,Sy2   Specify the top right corner of the source rectangle in source world co-ordinates (not required for *mode = 0*).

rop   The mixing function required.

Each plane of the target can be considered to be processed separately. For any pel in a target plane, three bits together with the *rop* values 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)
O   O   O              Index bit O   (LS)
O   O   1              Index bit 1
O   1   O              Index bit 2
O   1   1              Index bit 3
1   O   O              Index bit 4
1   O   1              Index bit 5
1   1   O              Index bit 6
1   1   1              Index bit 7   (MS)
```

The index formed in the above way determines the mixing required. *rop* actually contains not this index directly, but a long integer value which is an encoding for the operations required. A table of *rop* values for each value of the index can be found in an appendix. There are Mnemonic names are available for commonly-used mixes:-

```
ROP_SRCCOPY         0x000000CCL  /* SRC                          *
ROP_SRCPAINT        0x000000EEL  /* SRC OR DST                   *
ROP_SRCAND          0x00000088L  /* SRC AND DST                  *
ROP_SRCINVERT       0x00000066L  /* SRC XOR DST                  *
ROP_SRCERASE        0x00000044L  /* SRC AND NOT(DST)             *
ROP_NOTSRCCOPY      0x00000033L  /* NOT(SRC)                     *
ROP_NOTSRCERASE     0x00000011L  /* NOT(SRC) AND NOT(DST)        *
ROP_MERGECOPY       0x000000C0L  /* SRC AND PAT                  *
ROP_MERGEPAINT      0x000000BBL  /* NOT(SRC) OR DST              *
ROP_PATCOPY         0x000000F0L  /* PAT                          *
ROP_PATPAINT        0x000000FBL  /* NOT(SRC) OR PAT OR DST       *
ROP_PATINVERT       0x0000005AL  /* DST XOR PAT                  *
ROP_DSTINVERT       0x00000055L  /* NOT(DST)                     *
ROP_ZERO            0x00000000L  /* O                            *
ROP_ONE             0x000000FFL  /* 1                            *
```

mode   Specifies how eliminated lines/columns are treated if a compression is performed:-

BLTMODE_NOSCALE (0) - Do not stretch or compress
the data

BLTMODE_OR (1) - Stretch/compress as necessary,
OR'ing any eliminated rows/columns.
This is used for white on black.

BLTMODE_AND (2) - Stretch/compress as necessary,
AND'ing any eliminated rows/columns.
This is used for black on white

BLTMODE_IGNORE (3) - Stretch/compress as necessary,
ignoring any eliminated rows/columns.
This is used for color.

Other values of *mode* up to and including *32767* are
reserved. Values of *32768* and above may be used for
privately-supported modes for particular devices.

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

GPIERR_INVALID_SOURCE_GPI_HANDLE
GPIERR_INVALID_ARRAY_COUNT
GPIERR_INVALID_BITBLT_MIX
GPIERR_INVALID_BITBLT_MODE

GpiSetPel

```
SHORT GpiSetPel (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

This sets a pel, at a position specified in world co-ordinates,
using the current color.

If this function occurs while the drawing mode is store or draw-
and-store, order(s) will be constructed and placed in the current
segment.

Parameters:

hgpi   The handle of a Gpi presentation space.

x,y   Specify a position in world co-ordinates

Returns:

0 Error
1 OK
2 Correlate hit(s)

Principal errors:

-

GpiQueryPel

```
LONG GpiQueryPel (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

This returns the color of a pel, at a position specified in world co-ordinates.

Parameters:

hgpi   The handle of a Gpi presentation space.

x,y   Specify a position in world co-ordinates.

Returns:

-1 Error
>=0 Color index of the pel

Principal errors:

-

GpiFloodFill

```
SHORT GpiFloodFill (hgpi, bnd_color)
HPS hgpi;
LONG bnd_color;
```

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

The following current attributes of the presentation space are used:-

— Pattern color

— Pattern background color

— Pattern mix

— Pattern background mix

— Pattern set

— Pattern symbol

If this function occurs while the drawing mode is store or draw-and-store, order(s) will be constructed and placed in the current segment.

The results produced by this function are highly device dependent.

Parameters:

hgpi   The handle of a Gpi presentation space.

bnd_ color
            The color index which bounds the filled area.

Returns:

  0 Error
  1 OK
  2 Correlate hit(s)

Principal errors:

GPIERR_INVALID_BOUNDARY_COLOR

7.1.26   Region Support

7.1.26.1   Region Operations

Operations are provided to construct regions, which may range from simple rectangular shapes to complex shapes including ones with non-rectangular boundaries, or ones which are disjoint, or have holes in.

Once constructed, a region may be used for one of two purposes:

- To define a clipping region for a GPI presentation space. By default, the clipping region context extends over the whole drawing surface.

   Having defined (or defaulted) the clipping region, other functions are provided to change it.

- To draw a shape, defined by the region, into the presentation space.

As mentioned above, the generality of the functions allows complex regions to be generated. However, applications should use this capability with caution, since in some cases (especially in the case of clipping regions which are not simple rectangles) performance may be degraded.

Note that the clipping region described in these operations is the clipping region as perceived by an application. For actual drawing, this region has to be intersected with any clipping arising from the fact that the window on the device is of limited size, and may indeed be overlaid by other windows.

The co-ordinates with which regions are created and defined are taken to be device co-ordinates at the time a region is used to specify a clip window, or for drawing.

For clipping and other purposes, a point on the boundary of a rectangle or region is defined to be logically *inside* the rectangle or region. Any rectangle for which the right boundary is less than the left, or the top is less that the bottom, is deemed to be a NULL rectangle.

7.1.26.2   Uses for regions

A clipping region may be used to restrict drawing to a certain part of the picture which the application knows needs repainting, either because of its own operations, or because it is told by the system that a certain area needs repainting.

Care must be taken if metafiling or printing a picture constructed using clipping regions, since these are very device dependent. It is preferable where possible to restrict the use of clipping regions to 'healing' operations, and to use the other clipping functions defined in the section "Transform Functions".

A shape defined by a region can be useful for drawing complex areas, especially inverted shapes. However, again because of the device-dependent nature of clipping regions, it is preferable if possible to use the area functions defined in the section, "Area Functions".

7.1.26.3   GRECT and GPOINT structures

This section documents the GRECT and GPOINT data structures. They are:-

```
typedef struct
     LONG xl;
     LONG yb;
     LONG xr;
     LONG yt;
  GRECT;

typedef struct
     LONG x;
     LONG y;
  GPOINT;
```

7.1.26.4   Region Functions

GpiCreateRegion

```
        HRGN GpiCreateRegion (hdc, n, rects)
        HDC hdc;
        LONG n;
        GRECT rects[];
```

This creates a region, for a particular class of device, using a series of rectangles. The new region is defined by the OR of all of the rectangles specified.

Parameters:

hdc   The handle of a Device Context. A region suitable for
       use with the corresponding device is created.

n   The number of rectangles specified in *rects*. If $n = 0$,
       an empty region is created, and *rects* is ignored.

rects[] A long pointer to an array of *n* rectangles, with the
         data specified in the order *x1bl,y1bl,x1tr,y1tr,*
         x2bl,y2bl,x2tr,y2tr,..., where *x1bl,y1bl* are the co-
         ordinates of the bottom left of the first rectangle, and
         x1tr,y1tr are the co-ordinates of the top right of the
         first rectangle, etc.

Returns:

```
 0 Error
!=0 Region handle
```

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_ARRAY_COUNT
```

GpiSetRegion

```
BOOL GpiSetRegion (hdc, hrgn, n, rects)
HDC hdc;
HRGN hrgn;
LONG n;
GRECT rects[];
```

This is similar to GpiCreateRegion, except that it changes an
already existing region to be the OR of the supplied rectangles,
instead of creating a new region.

The previous contents of the region are irrelevant.

Parameters:

hdc   The handle of the Device Context for the device for
       which the region was created.

hrgn   A region handle.

n   The number of rectangles specified in *rects*. If $n = 0$,
       the region is set to EMPTY, and *rects* is ignored.

rects[] A long pointer to an array of *n* rectangles, with the
         data specified in the order *x1bl,y1bl,x1tr,y1tr,*
         x2bl,y2bl,x2tr,y2tr,..., where *x1bl,y1bl* are the co-
         ordinates of the bottom left of the first rectangle, and
         x1tr,y1tr are the co-ordinates of the top right of the
         first rectangle, etc.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE
GPIERR_INVALID_ARRAY_COUNT
```

GpiDestroyRegion

```
BOOL GpiDestroyRegion (hdc, hrgn)
HDC hdc;
HRGN hrgn;
```

This destroys a region.

Parameters:

hdc   The handle of the Device Context for the device for which the region was created.

hrgn   The handle of the region to be destroyed.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE
GPIERR_INVALID_OPERATION_FOR_CLIP_REGION
```

GpiCombineRegion

```
SHORT GpiCombineRegion (hdc, hrgn_dest, hrgn_src1,
      hrgn_src2, mode)
HDC hdc;
HRGN hrgn_dest;
HRGN hrgn_src1;
HRGN hrgn_src2;
LONG mode;
```

This combines two regions. The result is placed into the specified destination region, which may in fact be one of the two source regions.

Source and destination regions must all be of the same device class.

Parameters:

hdc   The handle of the Device Context for the device for which the region was created.

hrgn_ dest
      The handle of the destination region.

hrgn_src1,hrgn_src2
>The handles of the two regions to be combined.

mode   Method of combination, as follows:-

```
CRGN_OR       (1) - Union of src1 and src2
CRGN_COPY     (2) - src1 only (src2 ignored)
CRGN_XOR      (4) - Symmetric difference of
                    src1 and src2
CRGN_AND      (6) - Intersection of src1 and src2
CRGN_DIFF     (7) - src1 AND NOT(src2)
```

Returns:

```
0 Error
1 NULL region
2 RECTangular region
3 COMPLEX region
```

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE
GPIERR_INVALID_REGION_MIX
GPIERR_INVALID_OPERATION_FOR_CLIP_REGION
```

GpiEqualRegion

```
SHORT GpiEqualRegion (hdc, hrgn_src1, hrgn_src2)
HDC hdc;
HRGN hrgn_src1;
HRGN hrgn_src2;
```

This checks whether two regions are identical.

Both regions must be of the same device class.

Parameters:

hdc   The handle of the Device Context for the device for which the region was created.

hrgn_src1,hrgn_src2
>The handles of the two regions to be checked.

Returns:

```
0 Error
1 Not equal
2 Equal
```

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE
```

GpiOffsetRegion

```
BOOL GpiOffsetRegion (hdc, hrgn, x, y)
HDC hdc;
```

```
HRGN hrgn;
LONG x;
LONG y;
```

This moves the given region by the specified offsets.

Parameters:

hdc	The handle of the Device Context for the device for which the region was created.
hrgn	The handles of the region to be moved.
x,y	The increments by which the region is to be moved.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE
GPIERR_INVALID_OPERATION_FOR_CLIP_REGION
```

GpiPtInRegion

```
SHORT GpiPtInRegion (hdc, hrgn, x, y)
HDC hdc;
HRGN hrgn;
LONG x;
LONG y;
```

This checks whether a point lies within a region.

Parameters:

hdc	The handle of the Device Context for the device for which the region was created.
hrgn	The handle of the region.
x,y	The co-ordinates of the point.

Returns:

0 Error
1 Not in region
2 In region

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE
```

GpiRectInRegion

```
SHORT GpiRectInRegion (hdc, hrgn, rect)
HDC hdc;
```

```
HRGN hrgn;
GRECT rect;
```

This checks whether any part of a rectangle lies within the specified region.

Parameters:

hdc	The handle of the Device Context for the device for which the region was created.
hrgn	The handle of the region.
rect	A long pointer to a rectangle structure xbl,ybl,xtr,ytr, where xbl,ybl are the co-ordinates of the bottom left of the rectangle, and xtr,ytr are the co-ordinates of the top right of the rectangle.

Returns:

```
0 Error
1 Not in region
2 Some in region
3 All in region
```

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE
```

GpiQueryRegionBox

```
SHORT GpiQueryRegionBox (hdc, hrgn, rect)
HDC hdc;
HRGN hrgn;
GRECT rect;
```

Returns the dimensions of the tightest rectangle around the region, ie which completely encloses it.

If the region is null, the rectangle returned will have the right boundary less than the left, and the top boundary less than the bottom.

Parameters:

hdc	The handle of the Device Context for the device for which the region was created.
hrgn	The handle of the region.
rect	A long pointer to a rectangle structure in which are returned xbl,ybl,xtr,ytr, the co-ordinates of the bottom left and top right corners of the bounding rectangle.

Returns:

```
0 Error
1 NULL region
```

2 RECTangular region
3 COMPLEX region

Principal errors:

GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE

GpiQueryRegionRects

```
BOOL GpiQueryRegionRects (hdc, hrgn, bound, control, rects)
HDC hdc;
HRGN hrgn;
GRECT bound;
LPBUF control;
GRECT rects;
```

This returns the rectangles which, if OR'ed together, define the specified region.

Parameters:

hdc   The handle of the Device Context for the device for which the region was created.

hrgn   The handle of the region.

bound   A pointer to a bounding rectangle. Only rectangles which intersect this bounding rectangle will be returned. If this pointer is NULL, all rectangles in the region will be returned. If it is not NULL, then each of the rectangles returned will be the intersection of the bounding rectangle with a rectangle in the region.

control   A pointer to a structure which controls the processing. It contains the following:-

```
struct RegionRect {
    UINT start;
    UINT count;
    UINT retcount;
    UINT direction;
} RGNRECT;
```

start   The rectangle number to start enumerating at. Set this to zero to start from the beginning.

count   The number of rectangles which will fit into *xy_ array*. It must be at least 1.

retcount
          The number of rectangles actually written into *xy_ array*. A value below *start* indicates that there are no more rectangles to enumerate.

direction

The direction in which the (leading edge of the) rectangles are to be returned, as follows:-

```
RECTDIR_LFRT_TOPBOT  (1)  - left to right,
                          top to bottom
RECTDIR_RTLF_TOPBOT  (2)  - right to left,
                          top to bottom
RECTDIR_LFRT_BOTTOP  (3)  - left to right,
                          bottom to top
RECTDIR_RTLF_BOTTOP  (4)  - right to left,
                          bottom to top
```

rects   A long pointer to an array of rectangle structures, in which are returned $x1bl,y1bl,x1tr,y1tr,$ $x2bl,y2bl,x2tr,y2tr,...,$ where $x1bl,y1bl$ are the co-ordinates of the bottom left of the first rectangle, and $x1tr,y1tr$ are the co-ordinates of the top right of the first rectangle, etc.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_DC_HANDLE
GPIERR_INVALID_REGION_HANDLE
GPIERR_INVALID_REGION_CONTROL
```

7.1.26.5 Clipping Region Functions

A region may be selected for use as the clipping region of a Gpi presentation space. Since regions need not be rectangular, this may be used to provide non-rectangular clipping, though in a very device-dependent form. This clipping is independent from the clipping mandated by the layout of the windows on the screen (the VisRegion); ultimately, drawing is done to the inter-section of the application clipping region and the VisRegion.

To set up an application clipping region, a region of the required shape may first be constructed using the region functions described in the previous section. In this case, the co-ordinates used are device co-ordinates (device co-ordinates are the natural co-ordinates for the device, eg pels on a raster display; the origin is at the bottom left).

A particular region is established as the clipping region by GpiSelectClipRegion. Any further region operations (as defined in the previous section) on it are now invalid. However, the clipping region functions described in this section may now be

performed on it. These operate in *world co-ordinates,* and so are subject to current transformations before being combined with the existing clipping region.

When the clipping region is deselected, it reverts to being a normal region again. However, it retains the effect of any clipping region operations which were performed upon it.

A region may not be selected as the clipping region for more than one presentation space at a time.

GpiSelectClipRegion

```
HRGN GpiSelectClipRegion (hgpi, hrgn)
HPS hgpi;
HRGN hrgn;
```

Specifies the region to be used for clipping, when any drawing takes place through the specified presentation space.

Region operations (as opposed to clipping region operations) may no longer be performed upon the region, neither may it be selected into any other presentation space as a clipping region, until it becomes deselected again.

The co-ordinates of the region are taken to be device co-ordinates within the Device Context.

The previous clip region, if any, is converted to a region, and a handle to it is returned. This may be used in a subsequent GpiSelectClipRegion to reinstate the same clipping as before. If there was no clip region, a null handle is returned.

If this function occurs while the drawing mode is store or draw-and-store, order(s) will be constructed and placed in the current segment. Note, however, that this function is very device-dependent.

Parameters:

hgpi   The handle of a Gpi presentation space.

       The presentation space must be currently associated with a Device Context of the correct class of device (defined when the region was first created).

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

Returns:

```
-1 Error
 0 Null handle (no region was selected)
<-1 Old region handle
>0 Old region handle
```

Principal errors:

GPIERR_INVALID_REGION_HANDLE

GpiQueryClipRegion

```
HRGN GpiQueryClipRegion (hgpi)
HPS hgpi;
```

Returns the handle of the currently selected clip region.

The handle of the currently selected clip region, if any, is returned. If there is no currently selected clip region, a null handle is returned.

Parameters:

hgpi   The handle of a Gpi presentation space.

Returns:

```
-1 Error
 0 Null handle (no region was selected)
<-1 Old region handle
>0 Old region handle
```

Principal errors:

-

GpiQueryClipBox

```
SHORT GpiQueryClipBox (hgpi, rect)
HPS hgpi;
GRECT rect;
```

Returns the dimensions of the tightest rectangle which completely encloses the intersection of *all* of the various clipping definitions.

This includes the effect of all of the following:-

- Clip region
- Visible region (ie any windowing considerations)
- Clip area
- Viewing limits
- Graphics field

If the intersection is null, the rectangle returned will have the right boundary less than the left, and the top boundary less than the bottom.

Parameters:

hgpi   The handle of a Gpi presentation space.

rect   A long pointer to an array in which are returned *xbl,ybl,xtr,ytr*, the co-ordinates of the bottom left and top right corners of the bounding rectangle, in world co-ordinates.

Returns:

0 Error
1 NULL region
2 RECTangular region
3 COMPLEX region

Principal errors:

-

GpiIntersectClipRectangle

```
SHORT GpiIntersectClipRectangle (hgpi, rect)
HPS hgpi;
GRECT rect;
```

Sets the new clipping region to be the intersection of the current clip region and the specified rectangle.

If this function occurs while the drawing mode is store or draw-and-store, order(s) will be constructed and placed in the current segment.

Parameters:

hgpi   The handle of a Gpi presentation space.

rect   A long pointer to a rectangle structure, in which are returned *xbl,ybl,xtr,ytr*, where *xbl,ybl* are the co-ordinates of the bottom left of the rectangle, and *xtr,ytr* are the co-ordinates of the top right of the rectangle.

The co-ordinates are world co-ordinates.

Returns:

0 Error
1 NULL region
2 RECTangular region
3 COMPLEX region

Principal errors:

-

GpiExcludeClipRectangle

```
SHORT GpiExcludeClipRectangle (hgpi, rect)
HPS hgpi;
GRECT rect;
```

Excludes the specified rectangle from the clipping region.

A point on the boundary of the rectangle will be excluded from the clipping region.

If this function occurs while the drawing mode is store or draw-and-store, order(s) will be constructed and placed in the current segment.

Parameters:

hgpi   The handle of a Gpi presentation space.

rect   A long pointer to a rectangle structure, in which are returned *xbl,ybl,xtr,ytr*, where *xbl,ybl* are the co-ordinates of the bottom left of the rectangle, and *xtr,ytr* are the co-ordinates of the top right of the rectangle.

        The co-ordinates are world co-ordinates.

Returns:

```
0 Error
1 NULL region
2 RECTangular region
3 COMPLEX region
```

Principal errors:

-

GpiOffsetClipRegion

```
SHORT GpiOffsetClipRegion (hgpi, x, y)
HPS hgpi;
LONG x;
LONG y;
```

Moves the clipping region by the specified amounts.

If this function occurs while the drawing mode is store or draw-and-store, order(s) will be constructed and placed in the current segment.

Parameters:

hgpi   The handle of a Gpi presentation space.

x,y   The x and y increments in world co-ordinates by which the clipping region is to be moved.

Returns:

```
0 Error
1 NULL region
2 RECTangular region
3 COMPLEX region
```

Principal errors:

7.1.26.6 Drawing Functions

GpiPaintRegion

```
SHORT GpiPaintRegion (hgpi, hrgn)
HPS hgpi;
HRGN hrgn;
```

Paints the specified region into the specified presentation space, using the current Gpi pattern.

The current Gpi pattern foreground and background colours and mix and background mix of the presentation space are also used.

If this function occurs while the drawing mode is store or draw-and-store, order(s) will be constructed and placed in the current segment. Note, however, that this function is very device-dependent.

Parameters:

hgpi   The handle of a Gpi presentation space.

hrgn   The handle of the region.

Returns:

```
0 Error
1 OK
2 Correlate hit(s)
```

Principal errors:

```
GPIERR_INVALID_REGION_HANDLE
```

Chapter 8
Metafile Support

8.1 Metafile Support

A metafile is a sequence of functions which can cause all, or part, of a picture to be drawn. The functions which it contains are designed to retain as much as possible of the application's *intent*, so that the same picture may be reproduced on different systems as faithfully as possible. This concept is similar to the rationale for presentation spaces; indeed the information contained in graphics presentation spaces and metafiles is very similar.

Applications using metafiles may operate in the same workstation, and interchange metafiles using, for example, the clipboard, or they may be in other workstations reached via a network, or even on other machines.

Presentation Manager will support two levels of interchange. These are

- The metafile, which has the functionality of the Presentation Manager graphics, and
- Picture Interchange Format (PIF), which has lesser functionality.

Only the metafile form can be generated and played directly through the Presentation Manager graphics. A utility will be provided to convert between the metafile and PIF formats.

Only graphics functions are defined for a metafile. Vio data cannot be metafiled.

Certain raster operations will be included in a metafile, but their effect will be lost if the metafile is subsequently converted into a PIF interchange file. These are:-

- GpiBitBlt
- GpiSetPel
- GpiFloodFill
- GpiSelectClipRegion
- GpiOffsetClipRegion
- GpiIntersectClipRectangle
- GpiExcludeClipRectangle

The effect of these functions will be retained if the metafile is subsequently played through Presentation Manager, but because of their nature, unexpected effects may occur if they are replayed to a different device. Apart from differences arising from different pel resolutions, some functions may not work on some devices, for example, BitBlt operations on a plotter.

Note that GpiBitBlt operations are only recorded where the presentation space associated with the metafile is the *target* of the operation. If the source is a different presentation space, the size of the metafile may be large.

Any application colour table in force will be included in the metafile.

References to loaded symbol sets will be included in the metafile, as will logical fonts and linetype definitions. The actual character definitions, however, for both fonts and symbol sets, will not be metafiled.

The objective in recording a metafile is to permit the application to perform any sequence of (legal) operations, and on replay to be able to reproduce the same picture as that sequence would have generated if it had been issued directly at the same device context (assuming the same starting conditions).

Because the actual format of a metafile is a 'snapshot', this means that a certain amount of remapping must take place in order to generate an equivalent snapshot to the actual sequence of operations. For example, in non-stored mode, the same lcid may be re-used for different fonts. When such a sequence is metafiled, different lcids will be used by the system.

On replay, there is an additional requirement to be able to play a picture into a presentation space which already has objects in it. The origin of the picture in the metafile may be unknown, and it must therefore be presumed that it may use any legal values of segment names, lcids, etc, which might clash with those already in use in the presentation space. On replay, therefore, a similar process of remapping will take place to enable the metafile to be brought in as a 'subpicture'.

8.1.1   Presentation Manager Restrictions

The restrictions documented in this section override any statements elsewhere in this specification.

In the first release of Presentation Manager, there will be no remapping as described elsewhere, on either generate or replay. The latter means that only a limited subset of the MetPlayMetaFile options will be supported.

It will, however, still be possible to do the following:-

1. Generate a metafile in draw mode, and replay it also in draw mode, with the same initial conditions of the presentation space, and

2. Generate a metafile in any drawing mode, but subject to certain restrictions as described below, and replay it in any drawing mode, to an empty presentation space.

330

The restrictions, which only apply to case 2 above, are as follows:-

- Prior to the first draw operation to a presentation space with a metafile, the following must be established:-
  - Color table
  - Page window
  - Logical fonts
  - Symbol sets
  - Bitmaps for patterns
  - Line type definitions
  - Graphics field
  - Default viewing transform
- No changes to any of these must occur while the metafile Device Context is open.
- Once a bitmap has been used as the source of a GpiBitBlt operation, it must not be modified.

If these restrictions are followed (in both of the above cases this time), the interchange file generated will be one which can be converted to a PIF file (and can be used by the Plot Processor - see the chapter, "Spooler Interface").

On replay, the page units and the co-ordinate format must match those of the presentation space with which the metafile was generated.

A further restriction is that bitmaps may not be accesssed by any process other than the one which created/loaded it.

The following functions are not supported in the first release of the t.:-

- MetCopyMetaFile
- MetGetMetaFileBits
- MetSetMetaFileBits
- MetQueryMetaFileLength

The default code page of the presentation space will not be handled by remapping to a different default code page set in the first release of the Presentation Manager. It may either be ignored, or replace the default code page of the receiving presentation space.

The following restrictions apply to MetPlayMetaFile in the first release of the Presentation Manager:-

- No segment renaming will be done; *seg_base* must be 0 (or defaulted).
- No transform manipulation will be done; *load_type* must be 1 (or defaulted).
- *resolve* must be 1 (or defaulted).
- *lcids* must be either 1 (or defaulted) or 3. For 1, *no* lcid objects will be loaded, even if the lcid is not currently loaded.
- *line_type_sets* must be either 1 (or defaulted) or 3.
- *color_tables* must be either 1 (or defaulted) or 3.

All of the restrictions described in this section will be lifted in a subsequent release of the Presentation Manager.

8.1.2 Metafile Operations

At any time, a metafile may be in one of three states:-

1. In a file on disk. Filename is used to refer to it.
2. Known internally to the system, and kept in storage. Metafile handle used to refer to it.
3. Being written to within a Device Context. Only the DC handle is available to refer to it.

The intermediate state is provided so that if a metafile is being constantly used, it does not necessarily have to be fetched from disk every time. Similarly, if it has just been created, it may not need to be written to disk before being played. If there is insufficient storage, however, the system may need to use the disk file.

The functions are of two kinds:-

- Generation of a metafile, and
- Using an already generated metafile

If a metafile is either generated through a micro-PS, or played back through a micro-PS, then only the micro-PS function subset is allowed. In particular, stored segments will not be allowed, so no GpiCallSegment functions may be contained within it.

8.1.2.1 Metafile Generation

To create a metafile, a metafile Device Context must first be created, and a presentation space associated with it. Drawing into this presentation space then has the effect of putting data into the metafile. After all of the drawing has been completed, closing the Device Context closes the metafile, which is now in state 2 (above), ie there is a metafile handle available which may be used to access it. Once it is closed, no further drawing into it is possible; it may not be re-opened.

So long as the metafile Device Context remains open, multiple drawing operations, and even re-associations, are permitted.

After a metafile Device Context has been associated with a presentation space in one co-ordinate format, it cannot be associated with a presentation space which has a different co-ordinate format. On playback, however, it is not necessary for the presentation space to have the same format as that with which the metafile was created; conversion will be performed if necessary.

If one of the GpiDraw functions is issued, stored segments are 'drawn' into the metafile. A common operation would be to draw the graphics chain of segments into the metafile; this would create a metafile representing the current picture (assuming that the application had maintained a valid chain).

In draw mode, a (chained) metafile segment is created for each GpiOpen-Segment. It is not possible to tell from a metafile whether the graphics was originally stored or not.

Any graphics not introduced by a GpiOpenSegment (and therefore not stored), will be allocated a segment name of zero.

If there is more than one draw (stored or non-stored) then segments generated for draws after the first are added to the end of the chain. Applications are responsible for ensuring that duplicate segments are not added. This capability is provided so that applications which wish to may miss out parts of the chain, or use a mixture of stored and non-stored.

Dynamic segments will be metafiled if processed by one of the GpiDraw operations, and the 'draw dynamic segments' Draw Control (see GpiSet-DrawControl) flag is set. GpiRemoveDynamics and GpiDrawDynamics have no effect when the presentation space is associated with a metafile Device Context.

No regions except the Clip Region will be metafiled. On a GpiSelectClipRegion call, order(s) will be built into the metafile to describe the shape of the region. On playback, this clip region will replace the existing clip region. GpiIntersectClipRegion, GpiExcludeClipRegion, and GpiOffsetClipRegion will be metafiled as orders.

The graphics field defines the area of interest within the picture in the metafile (see the section, "Transform Functions").

The code page of the presentation space will be metafiled. On replay, if this is not the same as the code page of the receiving presentation space, a new default font will be set up with the code page of the metafile default set.

Escape functions will be metafiled.

Whether kerning is enabled will also be metafiled.

Segments are only metafiled if the display flag (see GpiSetDrawControl) is on. For primitives outside segments, the effect of changing the display flag will be recorded in the metafile, including the maintenance of current position and attributes across the change.

If a GpiErase function is performed to a presentation space associated with a metafile, the metafile data saved during the current association will be deleted. Close segment processing will be performed as usual for a Gpi-Erase.

The following sequence illustrates a simple example of generating a metafile:-

```
GpiCreatePS                  /* the presentation space    */
/* load fonts, color tables, etc                          */
DevOpenDC                    /* (type = metafile)         */
GpiAssociate                 /* the PS with the DC        */
GpiLine
   . . .                     /* draw the picture          */
GpiCloseDC                   /* returns metafile handle   */
MetSaveMetaFile              /* metafile now on disk      */
```

Query functions may be performed in the normal way to a Gpi presentation space associated with a metafile Device Context.

No correlation will be performed for a presentation space associated with a metafile Device Context.

8.1.2.2   Using a Previously-Generated Metafile

The operation of 'playing' a metafile into a presentation space causes the metafile contents to be drawn into the presentation space, and from there into any device for which a Device Context is currently associated with it.

Whether or not the metafile segments are stored in the presentation space depends upon whether the presentation space is in store (or draw-and-store) mode at the time the MetPlayMetaFile is issued (though any non-chained segments will be stored anyway). Note that in store mode, the

application will have to issue a draw function if it wishes the picture to be displayed.

If required, the application can issue GpiResetPS, before 'playing' the metafile.

The following sequence illustrates a simple example of playing a metafile:-

```
GpiCreatePS            /* the presentation space    */
DevOpenDC              /* the device to play it to   */
GpiAssociate           /* the PS with the DC         */
MetLoadMetaFile        /* get a metafile handle      */
MetPlayMetaFile        /* quote the metafile handle */
MetSaveMetaFile        /* no longer referring to it */
```

8.1.3   Uses for Metafiles

Metafiles can be used for the following purposes:-

- To interchange a picture with another application possibly at some remote node in the network.

- To generate a picture to be printed, again possibly at some remote node in the network. This might be used particularly where the eventual printer type was unknown.

- To save a picture for rapid recall and display, for example to show a series of pictures for a presentation.

- To record some common part of a picture, such as a company logo, which might be maintained centrally.

8.1.4   Functional Description

8.1.4.1   Metafile Interface Function Call Summary

Short list of the function calls:

- MetLoadMetaFile
- MetCopyMetaFile
- MetPlayMetaFile
- MetSaveMetaFile
- MetDeleteMetaFile
- MetGetMetaFileBits

- MetSetMetaFileBits
- MetQueryMetaFileLength

See also

- DevOpenDC
- DevCloseDC
- GpiAssociate

MetLoadMetaFile

```
HMF MetLoadMetaFile (hab, filename)
HAB hab;
LPSZ filename;
```

This loads a metafile, and returns a metafile handle which MetPlayMetaFile can use.

Parameters:

hab   The anchor block handle.

filename
          A string giving the metafile's filename.

Returns:

0 Error
!=0 Metafile handle

Principal errors:

```
GPIERR_INVALID_METAFILE_FILENAME
GPIERR_METAFILE_FILENAME_NOT_FOUND
```

MetCopyMetaFile

```
HMF MetCopyMetaFile (hmf_src)
HMF hmf_src;
```

Note: MetCopyMetaFile will not be supported in the first release of the Presentation Manager. See the section, "Presentation Manager Restrictions".

This copies a metafile. The source metafile must already have been loaded or generated, and is identified by a metafile handle. The new metafile is also identified by a handle, which is returned (it may be used, for example, by MetPlay-MetaFile).

Parameters:

hmf_src

The handle to the source metafile.

Returns:

0 Error
!=0 New metafile handle

Principal errors:

GPIERR_INVALID_METAFILE_HANDLE

MetPlayMetaFile

```
LONG MetPlayMetaFile (hgpi, hmf, count1, opt_array,
        count2, desc)
HPS hgpi;
HMF hmf;
LONG count1;
LONG opt_array;
LONG count2;
LPBUF desc;
```

Note: Only a restricted subset of MetPlayMetaFile will be supported in the first release of the Presentation Manager. None of the remapping functions will be supported. See the section, "Presentation Manager Restrictions".

This causes the specified metafile to be 'played' into the specified graphics presentation space.

Whether the graphics are drawn and/or stored in segment store depends upon the current drawing function mode (see GpiSetDrawingMode) in the presentation space (but note that any non-chained segments will always be stored). If chained segments are stored, they will be added to the end of any existing segment chain. A segment must not be open when this call is issued. At the completion of the call, there will be no open segment.

The application may need to reset the presentation space by GpiResetPS, before issuing this function.

Segments can be loaded either with the same identifiers which they had when the metafile was created, or with a new set of identifiers.

Segments retain the segment attributes which they originally possessed.

An attempt will always be made to load any symbol sets, logical fonts, and bitmaps used as shading patterns, providing that they use local identifiers (*lcid* s) which are not already in use. If an *lcid* is already in use, then the application may choose whether or not such objects are loaded. If they are loaded, then they will use currently unused *lcid* s, determined by the system. All references to these *lcid* s

within the segments will be mapped to the new values.

The option to load, and map *lcid* s, would normally be used by an application integrating a picture from another source, where no application or installation standards exist as to the use of *lcid* s. Where such standards do exist, the option not to load if the *lcid* is in use would be selected.

Parameters:

hgpi   The handle of the presentation space through which the metafile is to be played.

hmf   The handle of the metafile to be played.

count1   The number of elements specified in the *opt_ array* parameter.

opt_ array[count1]
    Various options which control the playing of the metafile:

seg_ base
        The starting segment identifier for storing segments.

        If either *seg_ base* or *count1* is zero, any segments which are stored retain their original identifiers.

        If *seg_ base* is greater than zero, the first named segment to be stored is given the identifier *seg_ base*; subsequent named segments are renumbered with consecutive segment identifiers. Any Call Segment order which references a segment has the segment identifier changed to the new identifier given to that segment.

        The default action for references which cannot be resolved is to warn the user that a called segment could not be found, and to ignore the call segment order. The *resolve* parameter below allows unresolved references to remain unchanged.

        Unnamed segments (those with a zero segment identifier) remain unnamed (that is, they retain a segment name of 0).

        If the current drawing mode is Draw, only unchained segments will be named. Chained segments will be drawn and

discarded.

load_ type

Specifies what transformations should be performed on the imported picture. The options are:

0   The default; same as 1.

1   The graphics are restored using the current window and viewport co-ordinate system (see GpiSetWindow and GpiSetViewport), rather than the one which was in use when the data was created. This is the default action.

2   The graphics field of the imported picture is drawn as large as possible within the current viewport, while preserving the aspect ration which the picture had when it was saved. The picture is centered, horizontally or vertically as appropriate, within the viewport. Any primitives which would have drawn outside the graphics field (but which would have been clipped) may or may not be clipped, depending upon the clipping currently in force.

3   The bottom left corner of the graphics field of the imported picture is placed at the origin (0,0) of model space, preserving the physical size of the picture when it was created.

same as 1

resolve Specifies the action to be taken when loading a metafile that contains call segment orders to segments which do not exist in the metafile. The options are:

0   Default; same as 1

1   The identifier of the called segment is not changed. It is presumed that the called segment is either already available

in segment store, or will be available when drawing takes place subsequently.

2   A warning is issued. This option is used if the metafile being loaded is expected to be complete.

lcids   Specifies the action to be taken for any

- Logical font definitions, or
- Symbol set mappings, or
- Bitmaps referenced by *lcid* s for use as shading patterns

which are held in the metafile, and for which the corresponding lcid is not already loaded.

The options are:

0   Default; same as 1

1   Do not load such objects. This is the default, and will be used where the application expects the correct objects to be already loaded.

2   Load such objects, and map the *lcid* s, including all references to it within the picture, to a new, currently unused, value. This would be used where the application has no consistent use of *lcid* s with the picture in the metafile.

3   Load all objects referenced in the metafile, first deleting any already existing in the presentation space, for which the referenced *lcid* is already in use.

line_ type_ sets
Specifies the action to be taken for any line-type definitions which are held in the metafile.

The options are:

0	Default; same as 1
1	Do not load. This is the default, and will be used where the application expects the correct definition to be already loaded.
2	Load the definitions with conflicting code point(s), changing all references to them within the picture, to new, currently unused, values. This would be used where the application has no consistent use of *line-type definitions* s with the picture in the metafile.
3	Unload any definitions already loaded in the presentation space, and load any definitions in the metafile.

color_ tables

Specifies the action to be taken with respect to any color table implied or present within the metafile.

The options are:

0	Default; same as 1
1	Ignore. The default or loaded color table in the presentation space is unchanged, as are the references to color attributes in the new data. This is the default; it is suitable where it is known that the currently loaded color table (if any) is suitable for the use of color in the imported picture.
2	Merge the color table implied or present in the metafile with the currently loaded one. Where the current color table already has an entry for a color in the metafile color table, color indices are remapped to that entry. Where one does not already exist, a new one is created and again indices are

remapped. This option would be used where maximum color fidelity for the existing picture, and the imported picture, is required.

3   Overwrite the currently loaded color table (if any), with that implied or present in the metafile. This could be used where there is no existing picture.

color_ realizable
Specifies whether the color realizable flag contained in the metafile should be used or not.

The options are:

0	Default; same as 1
1	Ignore the color realizable flag in the metafile. This is the default, and is used where the application playing the metafile will determine whether or not the color table should be realizable.
2	Set the presentation space's color realizable flag to the value of the flag in the metafile. This is used to ensure the same realizability as existed when the picture was saved.

kerning Specifies whether the global kerning enablement flag contained in the metafile should be used or not.

The options are:

0	Default; same as 1
1	Ignore the kerning enablement flag in the metafile. This is the default, and is used where kerning enablement is determined by the application playing the metafile.

2 Set the presentation space's kerning enablement flag to the value of the flag in the metafile. This is used to ensure the same kerning enablement state as existed when the picture was saved.

count2 The length supplied for *desc* parameter.

descriptor
The descriptive record, of up to 253 bytes, that was saved with the metafile.

Returns:

```
-1 Error
>=0 Number of segments which were renumbered.
   0 is always returned if either seg_base or
   count10.
```

Principal errors:

```
GPIERR_INVALID_METAFILE_HANDLE
GPIERR_INVALID_ARRAY_COUNT (count1)
GPIERR_INVALID_LENGTH (count2)
GPIERR_INVALID_PLAY_METAFILE_OPTIONS
GPIERR_SYMBOL_SET_DEFINITION_NOT_FOUND (warning)
GPIERR_LOGICAL_FONT_DEFINITION_NOT_FOUND
GPIERR_INCOMPATIBLE_METAFILE
GPIERR_STOP_DRAW_OCCURRED (warning)
```

MetSaveMetaFile

```
BOOL MetSaveMetaFile (hmf, filename)
HMF hmf;
LPSZ filename;
```

This deletes access to the specified memory metafile. The metafile itself is written to disk and may be accessed again by MetLoadMetaFile.

Parameters:

hmf The handle of the metafile which is to be saved.

filename
A string specifying the filename to which the metafile is to be written.

Returns:

```
0 Error
1 OK
```

Principal errors:

```
GPIERR_INVALID_METAFILE_HANDLE
```

GPIERR_INVALID_METAFILE_FILENAME

MetDeleteMetaFile

```
BOOL MetDeleteMetaFile (hmf)
HMF hmf;
```

This deletes access to the specified memory metafile. The memory metafile is destroyed.

Parameters:

hmf   The handle of the metafile which is to be deleted.

Returns:

0 Error
1 OK

Principal errors:

GPIERR_INVALID_METAFILE_HANDLE

MetGetMetaFileBits

```
BOOL MetGetMetaFileBits (hmf, offset, length, address)
HMF hmf;
LONG offset;
LONG length;
LPBUF address;
```

Note: MetGetMetaFileBits will not be supported in the first release of the Presentation Manager. See the section, "Presentation Manager Restrictions".

This transfers the specified metafile to application storage.

The total length of a metafile may be found from the data returned by MetQueryMetaFileLength. This function allows an application to retrieve the data is convenient sized units.

Parameters:

hmf   The handle to a memory metafile.

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

length The 4-byte length in bytes of the metafile data to copy.

address The address in application storage into which the metafile data is copied.

Returns:

0 Error

1 OK

Principal errors:

```
GPIERR_INVALID_METAFILE_HANDLE
GPIERR_INVALID_METAFILE_OFFSET
GPIERR_INVALID_METAFILE_LENGTH
```

MetSetMetaFileBits

```
BOOL MetSetMetaFileBits (hmf, offset, length, address)
HMF hmf;
LONG offset;
LONG length;
LPBUF address;
```

Note: MetSetMetaFileBits will not be supported in the first release of the Presentation Manager. See the section, "Presentation Manager Restrictions".

This transfers metafile data from application storage into a memory metafile Device Context. The application must ensure that the data is in the correct format - it should have been created by GetMetaFileBits, and not changed.

Parameters:

hmf	The handle to a memory metafile.
offset	The 4-byte offset in bytes into the metafile data from which the transer must start. This is used when the metafile data is too long to fit into a single application buffer.
length	The 4-byte length in bytes of the metafile data to copy.
address	The address in application storage from which the metafile data is to be copied.

Returns:

0 Error
1 OK

Principal errors:

```
GPIERR_INVALID_METAFILE_HANDLE
GPIERR_INVALID_METAFILE_OFFSET
GPIERR_INVALID_METAFILE_LENGTH
```

MetQueryMetaFileLength

```
LONG MetQueryMetaFileLength (hmf)
HMF hmf;
```

Note: MetQueryMetaFileLength will not be supported in the first release of the Presentation Manager. See the section, "Presentation Manager Restrictions".

This returns the total length in bytes of a memory metafile. This is for use in subsequent MetGetMetaFileBits functions.

Parameters:

hmf   The handle to a memory metafile.

Returns:

-1 Error
>=0 Total length of the metafile

Principal errors:

GPIERR_INVALID_METAFILE_HANDLE

Chapter 9
Advanced Vio Interface

9.1 Advanced Vio Interface

9.1.1 Extensions for Advanced Vio

For Presentation Manager the DOS base Vio interface is extended to provide the following additional functions:

- support for a Device Context environment
- support for multiple presentation spaces
- support for multiple character sets
- support for extended attributes

The new interface is called Advanced Vio.

9.1.1.1 Advanced Vio presentation spaces

An Advanced Vio presentation space is created by VioCreatePS. This may have 2 or 4 bytes per character cell. Shown below are representations of the two supported presentation space formats.

2 bytes per cell (CGA format)

```
*--------------------------------------------------------
| |            |               | |            |           | |
| | code       |    CGA        | | code       |    CGA    | |...
| | point      | attribute     | | point      | attribute | |
| |            |               | |            |           | |
*--------------------------------------------------------
   <---------------------->
            cell
```
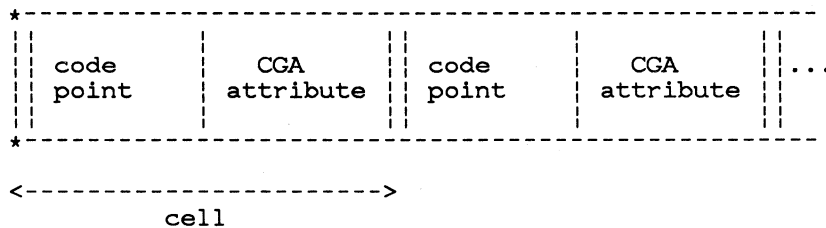
4 bytes per cell (extended format)

```
*-----------------------------------------------------------------
| |          |           |           |          | |          |
| | code     |    CGA    | extended  |  spare   | | code     |...
| | point    | attribute | attribute |          | | point    |
| |          |           |           |          | |          |
*-----------------------------------------------------------------
   <------------------------------------------------>
                        cell
```
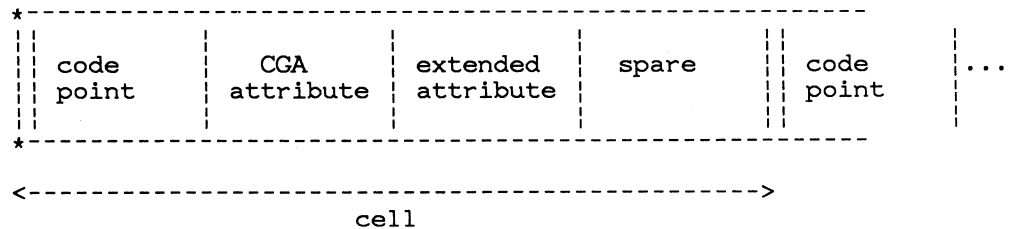
Attribute byte contents.

— CGA Attribute byte

```
Bits 7-4 - background colour
Bits 3-0 - foreground colour
```

Note Presentation Manager display device drivers do not support
the blink or intensify attributes.

— Extended Attribute byte

```
Bit 7     - underscore
Bit 6     - reverse video
Bit 5     - reserved
Bit 4     - background transparency (transparent=1, opaque=0)
Bits 3-2 - reserved
Bits 1-0 - character set 0,1,2 or 3
```

Note See section "Advanced Graphics and Alphanumerics" for
details of transparency support in Presentation Manager.

— Spare Attribute byte

```
for application use
```

If one attribute byte is present, then the system assumes suitable defaults
(eg no highlighting, opaque, character set 0).

The actual colours that appear on the screen will depend on the current
physical colour palette. It is the application's responsibility to ensure the
colours displayed are the colours expected.

All valid Vio function calls may access all Advanced Vio presentation
spaces. It is up to the caller to ensure that data is in the correct format
for a particular function, i.e. VioWrtCellStr to a 3 attribute byte presenta-
tion space should have cells 4 bytes long.

9.1.1.2 Multiple Windows/Bitmaps

An application can create and use a number of screen Windows and Bit-
maps. In order to draw into one of these using an Advanced Vio presenta-
tion space, the latter must first be associated with the appropriate Device
Context. A presentation space may be associated with only one Device
Context at a time. An association to another Device Context will cause an
implicit dissociation from the current association.

If an Advanced Vio presentation space is associated with a metafile Device
Context, the metafile data will be converted to graphics format.

9.1.1.3 Character sets

An application can reference up to 4 character sets, 1 base and 3 loadable. All loadable character sets are image character sets. The character set to be used for a particular character is controlled by the character set bits, where character set 0 is the base character set.

The base character set to be used for an Advanced Vio presentation space is set when the presentation space is created and will be ASCII. Presentation Manager will issue DosGetCp to determine which ASCII code page is the default for the process and will use that for the base character set.

The Vio code page function calls have been extended to allow for switching to the EBCDIC CECP supplied with Presentation Manager.

An application that wishes to use an EBCDIC base character set is required to issue a VioSetCp to switch the base character set for the created presentation space. All base character set definitions for all supported device cell sizes are supplied in the Presentation Manager screen group.

The default Vio presentation space may not be switched from an ASCII base character set.

The application can define its own character set definitions. The VioGet-DeviceCellSize function will return the current device cell size in pels so that character sets can be specified in the correct format. An application can copy its own character set definition to any of the loadable character sets.

Unpredictable results will occur if a loadable character set is used before a character set definition has been loaded.

Only symbol sets may be used with Advanced Vio. Fonts (see the section, "Character Functions" in the chapter, "Graphics Programming Interface") are not available.

9.1.1.4 Device Cell Size Support

There is support in Presentation Manager to allow an application to use any device cell size a particular device supports. The device cell size is specified in pels. There is a default device cell size for a particular device. This cell size will be used when a presentation space is associated with a Device Context.

An Advanced Vio application may alter the device cell size by issuing VioSetDeviceCellSize specifying the required cell size in pels. Presentation Manager will best fit the required size to one of those supported by the particular device. The application should issue VioGetDeviceCellSize to

check what cell size was set. The device cell size is chosen such that for a given supported device cell the cell width is less than or equal to the required cell width AND the cell height is less than or equal to the required cell height. If no supported cell size meets these criteria, then the smallest supported cell size will be selected.

A default Vio application (see the section, "Support of MS OS/2 VIO, KBD, and MOU Calls") may vary the device cell size by altering the number of Alpha Rows with VioSetMode. The number of rows requested must be supported by the particular device.

When the device cell size is altered Presentation Manager will switch to using the base character set definition corresponding to the cell size. It is an applications responsibility to ensure that loaded character set definitions are valid. If a mismatch exists between the cell size specified in the Presentation Space and the loaded symbol set, then the characters defined in the symbol sets will appear as blanks.

9.1.1.5 Advanced Vio Query functions

Two query functions are supplied for an application. VioQueryPSFormats will return a list of presentation space formats supported in the Presentation Manager screen group. VioQuerySymbolSets will return a list containing the current base character set plus all loaded character sets for an Advanced Vio presentation space.

DevQueryCaps is extended to indicate whether windows used for Advanced Vio on a particular device

- Must be character aligned,
- Should be character aligned for best performance, or
- May be either character or pel aligned, with no significant impact on performance.

Undefined results are produced for devices which return the first option, if this is not adhered to.

9.1.1.6 Screen content

The size of the Advanced Vio presentation space is specified by the application. If it is smaller than the window in either direction, then the excess area is left unchanged; if larger, then the excess data is not displayed.

The origin (ie data to be displayed in the top left of the window) may be specified by the application. If an area of the window is "exposed" by the origin being altered such that the presentation space right or bottom edge moves left or up in the window then the "exposed" area is cleared. This

would also remove any graphics present in the area. The application would need to restore the graphics as documented in section "Advanced Graphics and Alphanumerics".

After directly updating the Advanced Vio presentation space the application issues a call to request the system to reflect the change on the screen. This function is already available for DOS base Vio via the VioShowBuf function call; however it is extended in Advanced Vio to allow the application to specify the rectangle within the presentation space which bounds the changes made since the last such call. The system may optionally optimise its performance by only updating from within this rectangle. This new function is achieved using the VioShowPS function call.

9.1.1.7 WM_ SIZE Message Processing

Avio applications must pass the processing for WM_ SIZE messages to the routine WinDefAVioWindowProc:

```
ULONG FAR PASCAL WinDefAVioWindowProc(lpVioPS, hWnd, wMsg,
                                        lParam1, lParam2);
HPS      FAR * lpVioPS;
HWND     hWnd;
UINT     wMsg;
ULONG    lParam1;
ULONG    lParam2;
```

where hWnd, wMsg, lParam1, and lParam2 are taken from the original message arguments, and lpVioPS is a long pointer to the AVio presentation space associated with the designated window.

This routine, which only handles WM_ SIZE messages, maintains the size data in the presentation space. It must be called before the application attempts to access the window.

9.1.2 Restrictions to DOS base Vio

Any restrictions to DOS base Vio are listed below, either as restrictions to supported DOS base calls or by the call being unsupported.

9.1.3 DOS base Vio - Functional Description

All DOS base Vio functions calls are supported in the Presentation Manager screen group although certain calls are invalid and will return error return codes if issued.

9.1.3.1 Valid Function call summary

The following function calls are handled by Presentation Manager. All may use a zero or non-zero Vio handle (unless otherwise stated). Calls with a zero handle will be routed to the default presentation space (LVB) and default window.

VioEndPopUp
> Deallocate a popup display screen. Only valid for default Vio. An error will be returned if issued with a non zero handle.

VioGetAnsi
> Get Ansi state.

VioGetBuf
> This function returns the address and length of the Advanced Vio presentation space. The presentation space may be used to directly manipulate displayed information.

VioGetConfig
> Get the video configuration. Only adapter type and display type are set.

VioGetCp
> Query code page currently used to display text on the screen.

VioGetCurPos
> This function returns the current row and column position of the cursor.

VioGetCurType
> This function returns the cursor type. The cursor type consists of the cursor start line, end line, width (always returned as 0 - one column width) and attribute (normal or hidden).

VioGetMode
> This function returns the current mode of the video display. Only valid for default Vio. An error will be returned if issued with a non zero handle.

VioPopUp
> Allocate a popup display screen. Only valid for default Vio. An error will be returned if issued with a non zero handle.

VioPrtSc
> Print the contents of the screen.

VioPrtScToggle
> Toggle the Ctrl-PrtSc flag.

VioReadCellStr
> Read a string of character/attributes (or cells) from the Advanced Vio presentation space starting at the specified

location.

VioReadCharStr

Read a character string from the Advanced Vio presentation space starting at the specified location.

VioScrollDn

Scroll the current screen down the specified number of lines.

VioScrollLf

Scroll the current screen left the specified number of columns.

VioScrollRt

Scroll the current screen right the specified number of columns.

VioScrollUp

Scroll the current screen up the specified number of lines.

VioSetAnsi

Set Ansi on or off.

VioSetCp

Set code page currently used to display text on the screen.

VioSetCurPos

Position the cursor to the specified row and column on the display.

VioSetCurType

Set the cursor type. The cursor type consists of the cursor start line, end line, width (assumed 0 - one column width) and attribute (normal or hidden).

VioSetMode

This function sets the mode of the video display. Only valid for default Vio. An error will be returned if issued with a non zero handle. The only variable that will be examined is the Alpha Rows. This must valid for the particular device.

VioShowBuf

This function updates the display with the Advanced Vio presentation space. The caller may specify a subset of the presentation space for update.

VioWrtCellStr

Write a character attribute string to the Advanced Vio presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

VioWrtCharStr

Write a character string to the Advanced Vio presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

VioWrtCharStrAtt
> Write a character string with a repeated attribute code to the Advanced Vio presentation space. The caller must specify the starting location on the presentation space where the string is to be written.

VioWrtNAttr
> Write an attribute code to the Advanced Vio presentation space a specified number of times. The caller must specify the starting location on the presentation space where the attribute code is to be written.

VioWrtNCell
> Write a cell (or character,attribute) to the Advanced Vio presentation space a specified number of times. The caller must specify the starting location on the presentation space where the cell is to be written.

VioWrtNChar
> Write a character to the Advanced Vio presentation space a specified number of times. The caller must specify the starting location on the presentation space where the character is to be written.

VioWrtTTY
> Write a character string from the current cursor position in TTY mode to the Advanced Vio presentation space. The cursor will be positioned at the end of the string+1 at the end of the write.

9.1.3.2 Invalid Function call summary

The following function calls must not be issued by an application. An error code will be returned if any calls are issued.

VioDeRegister
> Deregister a video subsystem

VioGetFont
> Return the current font.

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.

VioSetFont
> Set the display font.

VioSetState
> Set the video state.

9.1.4 Advanced Vio - Functional Description

9.1.4.1 Function call summary

The following DOS Advanced Vio functions have been defined:-

VioAssociate
> Associate an Advanced Vio presentation space with a Device Context

VioCreatePS
> Allocate an Advanced Vio presentation space

VioDeleteSymbolSet
> Delete a symbol set

VioDestroyPS
> Destroy the Advanced Vio presentation space

VioGetDeviceCellSize
> Get the current device cell size

VioGetOrg
> Get Origin

VioLoadSymbolSet
> Load a Symbol Set

VioQueryPSFormats
> Query supported Advanced Vio presentation space formats.

VioQuerySymbolSets
> Query base symbol set plus all loaded symbol sets.

VioSetDeviceCellSize
> Set the device cell size

VioSetOrg
> Set Origin

VioShowPS
> Update the display with the Advanced Vio presentation space for a rectangle

9.1.4.2 Function call detail

VioAssociate (PresH,DCH)

> Associate an Advanced Vio presentation space with a Device Context. Any type of Device Context may be used. Subsequent VioShowPS or VioShowBuf functions direct output to this Device Context.

> If the Advanced Vio presentation space is currently associated with another Device Context, that association will be broken. Similarly, if another Advanced Vio presentation space is currently associated with the Device Context, that association is broken.

> A screen Device Context is the only kind of Device Context which may be associated with an Advanced Vio presentation space.

> If a null handle is supplied for the Device Context, the presentation space is just dissociated from the currently associated Device Context.

> Parameters:

> PresH (type hpvs)
>> The presentation space handle returned to the application by VioCreatePS.

> DCH   The Device Context handle. If null, a dissociation occurs.

> Returns:

> *0* Error
> *1* OK

VioCreatePS (Reserved,NumAttr,Format,Width,Depth,PresH_ var)

Allocate an Advanced Vio presentation space.

Parameters:

Reserved

Reserved (must be zero).

NumAttr

The number of attribute bytes in the presentation space per character cell. This may be 1 or 3.

Format   Identifies the format of the attribute byte(s) in the presentation space. Currently Format 0 is the only defined format; for this, the presentation space layout is:

```
NumAttr = 1  (CGA)
   Codepoint
   Base attributes     background colour
                       foreground colour
                       (blink and intensify are
                        not supported)

NumAttr = 3  (Extended)
   Codepoint
   Base attributes     background colour
                       foreground colour
                       (blink and intensify are
                        not supported)
   Extended attrs      underscore
                       reverse video
                       background opacity
                       character set 0,1,2 or 3
   Spare byte          for application use
```

Width,Depth

Specify the size of the presentation space required in character cell units. *Note:*
The presentation space size (Width $*$ Depth $*$ (NumAttr $+$ 1)) must not exceed 32k.

PresH_ var (type hpvs)

The variable in which the presentation space handle is returned to the application, which must be passed on all subsequent Vio,Advanced Vio calls for this presentation space.

Note that this is a 16-bit handle, unlike other Presentation Manager handles.

Returns:

0 Error
1 OK

VioDeleteSymbolSet (PresH,Rsvd,Number)

> Delete a Symbol Set

> Parameters:

> PresH (type hpvs)
>> The presentation space handle returned to the
>> application by VioCreatePS.

> Rsvd   Reserved (must be 0).

> Number
>> The number (1,2 or 3) of the loadable symbol set
>> to be freed.

> Returns:

>> *0* Error
>> *1* OK

VioDestroyPS (PresH)

> Destroy the Advanced Vio presentation space.

> Parameters:

> PresH (type hpvs)
>> The presentation space handle returned to the
>> application by VioCreatePS.

> Returns:

>> *0* Error
>> *1* OK

VioGetDeviceCellSize (PresH,Width,Height)

> Get current device cell size

> Parameters:

> PresH (type hpvs)
>> The presentation space handle returned to the
>> application by VioCreatePS.

> Height,Width
>> The current device cell height and width in pels.

> Returns:

>> *0* Error
>> *1* OK

VioGetOrg (PresH,Column,Row)

> Get Origin

> Parameters:

PresH (type hpvs)
>The presentation space handle returned to the application by VioCreatePS.

Row,Column
>Identifies the cell currently mapped to the top left hand corner of the window (first is zero).

Returns:

0 Error
1 OK

VioLoadSymbolSet (PresH,Rsvd,Number,Name,Length,DataAddr)

Load a Symbol Set definition from a specified application data area. This is the Vio equivalent of GpiLoadSymbolSet. See the section, "Character Functions" in the chapter, "Graphics Programming Interface" for more details.

Parameters:

PresH (type hpvs)
>The presentation space handle returned to the application by VioCreatePS.

Rsvd   Reserved (must be 0).

Number
>The number (1,2 or 3) of the loadable symbol set being defined.

Name   An 8 character name which may be used to describe the symbol set, for interchange for example.

Length   Specifies the length of the data. This should be the same as the length field at the start of the data.

DataAddr
>Specifies the start address of the data area which contains the symbol set definition. Only row loaded image symbol set definitions are supported for Advanced Vio. The definition should match the character cell size.

Returns:

0 Error
1 OK

VioQueryPSFormats (Reserved,QueryFormats)

Query supported Advanced Vio presentation space formats.

Parameters:

Reserved

Reserved (must be zero).

QueryFormats

A structure into which a list of presentation space formats supported by Presentation Manager is returned.

Returns:

0 Error
1 OK

VioQuerySymbolSets (PresH,Length,DataAddr)

Query the base symbol set plus all loaded symbol sets. This is the Vio equivalent of GpiQuerySymbolSets. See the section, "Character Functions" in the chapter, "Graphics Programming Interface" for more details.

Parameters:

PresH (type hpvs)

The presentation space handle returned to the application by VioCreatePS.

Length   The length of the data buffer provided.

DataAddr

The address of a data buffer in which to return the information.

Returns:

0 Error
1 OK

VioSetDeviceCellSize (PresH,Width,Height)

Set the device cell size

Parameters:

PresH (type hpvs)

The presentation space handle returned to the application by VioCreatePS.

Height,Width

The required device cell height and width in pels.

Returns:

0 Error
1 OK

VioSetOrg (PresH,Column,Row)

> Set Origin
>
> Parameters:
>
> ---
>
> PresH (type hpvs)
>> The presentation space handle returned to the application by VioCreatePS.
>
> Row,Column
>> Identifies the cell to be mapped to the top left hand corner of the window (first is zero).
>
> Returns:
>
> *0* Error
> *1* OK

VioShowPS (PresH,Offset,Width,Depth)

> Update the display with the Advanced Vio presentation space for a rectangle
>
> Parameters:
>
> ---
>
> PresH (type hpvs)
>> The presentation space handle returned to the application by VioCreatePS.
>
> Offset   Identifies the cell of the top left hand corner of the rectangle which has been updated since the last VioShowPS. The top left cell is offset 0.
>
> Width,Depth
>> Defines the dimensions of the rectangle in character cell units.
>
> Returns:
>
> *0* Error
> *1* OK

Chapter 10
Standard Application Support

10.1 Support of DOS VIO.., KBD.. and MOU.. calls

Applications that are written to make use of a subset of the DOS VIO, KBD, and MOU api calls may be run in the Presentation Manager screen group. Appendix B describes the limitations that must be followed if an application is to be run in the Presentation Manager screen group. This section describes the support for those applications that follow those limitations.

The objective of this support is to allow a significant proportion of DOS applications to run without modification in a window of the Presentation Manager screen group. That is, those applications should be able to run in a Presentation Manager window, but they will not have to make any calls except the DOS VIO, KBD, and MOU calls.

Such an application must be identified to Presentation Manager at installation time. Failure to do so will result in an error code being returned if a DOS VIO, KBD or MOU call is executed in an application.

The calls supported are those specified in DOS level 1.1 specification with the restrictions as noted below.

10.1.1 Default window and presentation space

A default window and presentation space will be created for the application. The effect of this will be that an environment will be created for applications that will allow them to run as if they were in their own screen group.

More details follow.

10.1.1.1 Default window creation

A default window will be created when a DOS VIO, KBD or MOU application is loaded. One default window is created for each of these applications. This window will be shared for every process of the application. However, two separate applications will each receive their own default window. This mirrors the situation that would pertain if each application were to be run in its own screen group.

10.1.1.2 Default window characteristics

The default window will have the following characteristics

— The window will have normal borders

— The window will have a title bar

— The window will be sizeable by the user interface

— The window will be moveable by the user interface

— The window will be maximisable

— If the window requires scrolling it will contain scroll bars

— The window will initially occupy the minimum of:

— The entire screen

— The space required to allow an 80 x 25 character client area

10.1.1.3 Default presentation space

A default presentation space will be created at the same time as the default window. This presentation space will be associated with the default window. The presentation space will have the following characteristics:

— The size will be 80*25

— The format will be CGA format

The size of the presentation space will alter to reflect any valid settings of the number of rows specified in VioSetMode.

10.1.2 VIO calls

Any VIO call from the application that specifies a zero handle will be directed to the default presentation space and window. The VIOGETBUF call will return the address of the default presentation space, and this is in the same format as the DOS logical video buffer. See the section, "DOS Base Vio - Functional Desription", for the restrictions on Vio calls.

The application may also issue the following extended VIO calls to the default window.

— VioGetOrg

— VioSetOrg

— VioQueryPSFormats

— VioQuerySymbolSets

— VioShowPS

If it does so, however, it will be unable to run in its own screen group, or in a system without Presentation Manager

10.1.3 KBD calls

The application may issue KBD calls to a default keyboard. A default keyboard will exist for each application. All processes in an application will share the same default keyboard. Logical keyboards are not supported by Presentation Manager.

10.1.3.1 Restricted Function KBD calls

The following KBD calls have a modified effect when directed to the default window.

KbdSetStatus
> Only Binary/ASCII, Turnaround character and Echo state may be modified with this call. All other functions are no-oped.

KbdGetStatus
> The data returned by this call is as follows:
>
> — Word 0: length=5
>
> — Word 1: Echo & input mode as per DOS
>
> — Word 2: Turnaround character as per DOS
>
> — Word 3: Interim character flags always 0
>
> — Word 4: Shift state as per last character enqueued into keyboard buffer.

10.1.3.2 Invalid KBD calls

The following KBD calls must not be issued by an application. An error code will be returned if any calls are issued.

KbdRegister
> Register keyboard subsystem

KbdDeRegister
>   Deregister keyboard subsystem

10.1.4   MOU calls

The application may issue MOU calls. A MOU call will receive input from
the locator when the default window is the input window.

10.1.4.1   Restricted Function MOU calls

The following MOU calls have a modified effect when directed to the
default window.

MouOpen
>   The handle returned by MouOpen will be unique for the
>   application, not the screen group as in base DOS. This han-
>   dle must be used in future MOU calls.

MouClose
>   This call closes the locator device for the current application.
>   After this call the handle is invalid.

MouSetPtrShape
>   The MOU support is limited to text mode emulation, and
>   application defined pointers for these modes will not be
>   displayed by Presentation Manager. The Presentation
>   Manager system default pointer will be displayed in the
>   default window in this case. The pointer shape therefore has
>   no meaning to Presentation Manager but will be returned by
>   a subsequent MouGetPtrShape call.

MouGetPtrShape
>   This call will work the same as in base DOS with the limita-
>   tion that the pointer shape is always the text pointer as
>   defined by the MouSetPtrShape call for the application.

MouGetPtrPos
>   This call will return the position of the Presentation
>   Manager pointer (in character units) when the pointer is
>   within the default window. If the pointer is not in the
>   default window at the time the call is made, an error will be
>   returned.

MouGetScaleFact
>   This call will always return the current scaling factor as set
>   by Presentation Manager, namely a row and column scaling
>   factor of 1 character.

MouGetDevStatus
>   This call will always return a word of 0.

MouSetPtrPos
>   This call will set the pointer position. If the pointer position is being set to a position that would not be visible, or if the vio window is not the active window, then the pointer position is not set and an error is returned. Otherwise, the pointer is moved to the relevant point on the screen.

MouDrawPtr
>   The Presentation Manager system pointer will always be displayed by Presentation Manager when the pointer is within the default window. Presentation Manager treats this call as a noop, and returns a zero return code.

MouRemovePtr
>   Collision avoidance is provided automatically by Presentation Manager and application control is redundant. The pointer will always be on the screen (if a mouse is attached to the system), and the application is unable to remove it with this call. Presentation Manager treats this call as a noop, and returns a zero return code.

MouSetScaleFact
>   Applications are not able to change the scaling factor within the Presentation Manager screen group. Presentation Manager treats this call as a noop, and returns a zero return code.

MouSetHotKey
>   Presentation Manager will always return ERROR-MOUSECANT_ RESET to this call.

10.1.4.2 Invalid MOU calls

The following MOU calls are invalid when directed to the default window.

MouRegister
>   Register a mouse subsystem

MouDeRegister
>   Deregister a mouse subsystem

10.1.5   Other Presentation Manager calls

No other Presentation Manager api calls are allowed to the default window or presentation space.

However, other Presentation Manager calls may be made within the application to other windows, presentation spaces, etc.. In order to use the other Presentation Manager functions, the application *will have to create an additional input queue* for the use of these other functions.

Chapter 11
Spooler Interface

11.1 The Spooler

11.1.1 Introduction

The Spooler intercepts data going to printer Device Drivers, and writes it to disk. The data is then printed when it is complete, and the required printer is free. This prevents the printer output from different sources being intermixed. *Note:* The generic term *print* is used to describe both printing on a printer device and plotting on a plotter device. Similarly, *printer* refers to either a printer device or a plotter device. The printer device class is used to determine the device type (eg raster printer, vector plotter).

11.1.2 Overview

```
                                          +---------+
                                          |User I/F |<+
                 (10)                     +---------% |(11)
       +----------------------------+       |     |
       |                      (3)    |       |  +---%
       |     +--------------------+  |       V  V V
  +-------+(1)+-+                 |  |    +--------+              +-+
  |Appl   |-->| |                 |  |    |        |             | |
  |       |<->| |                 |  |    |        |             | |
  +------%(9) | |                 |  |    |        |             | |
       |A| (2)+---------+(3)      |  |    |        |          |A| (6)+---------+
       |P|-->|Printer  |-->       |  |    |********|(4)+------+(5)|P|-->|Printer |-+
       |I|<->|DD-I     |          |  |    |*LPT1  *--->|Print|-->|I|   |DD-II   | |
       | |(9)+---------%          |  |    |********|   |Queue|   | |   +--------% |
       | |                        |  |    |        |   |Proc |   | |     (15)    |
       | |         +-------------->|Queue |   +-----%   | |             |
       | |         |(17)   +->|Manager|             |A|            |
       |+-%        |       |   +-------%             +-%            |
    +----------%   +----------+ (14)|                               |
  +----------+   +----------+   |                                   |
  |DOS       |   |DOS       |-% (16)      +--------+          |  (7) |
  |PRINT     |   |SPOOL     |<--------------|DD      |<--------%     |
  |Command   |   |Command   |<+             |Register|                |
  +---------%   +---------%  |             +--------%                |
       |            |        |                                       |
       |            |    (13)|       +----------------------------------------%
       |            |        |       |
       |            |        V
       |            |     +--------+
  +-------+ +---------------->|DOS     |
  |DOS   |-|---------------->|Printer |
  |Appl  | |   (12)          |DD      |
  +-----%          +-------%
                               |(8)
                               V
                            +--------+
                            |Printer |
                            +--------%
```
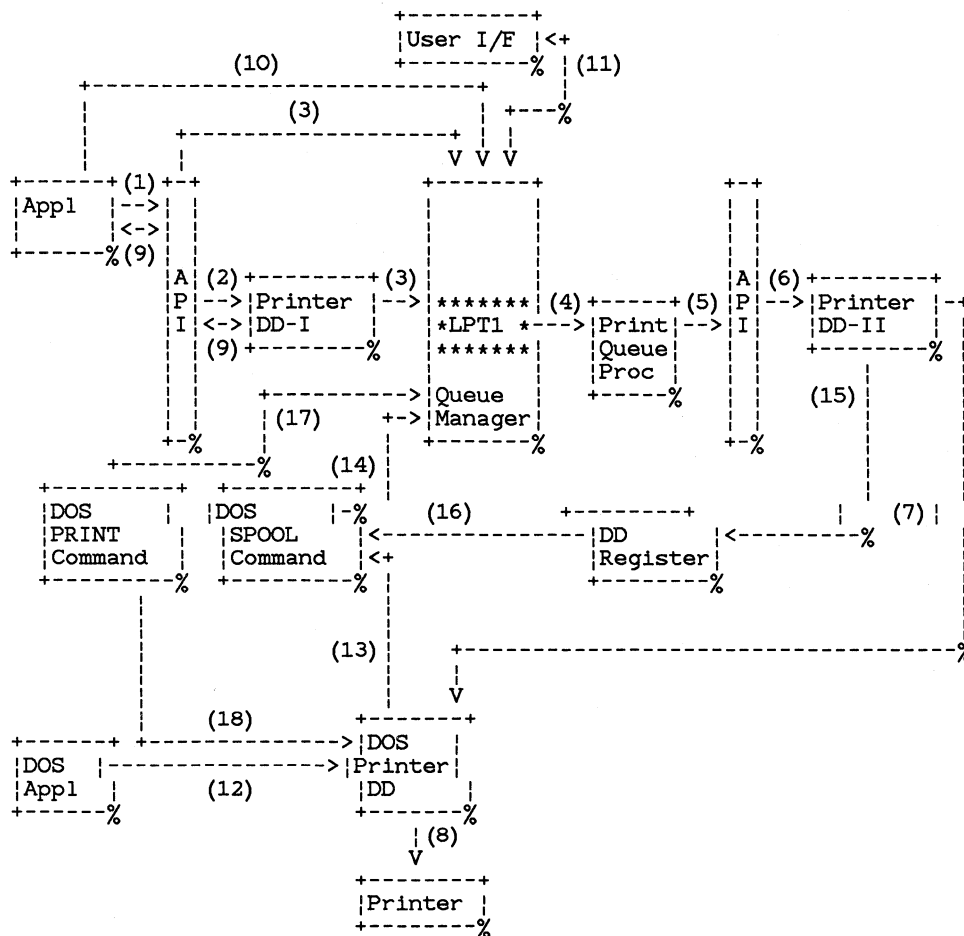
Figure 11.1 Spooler Logical Data Flow

The Presentation Manager Spooler replaces the DOS Spooler, and is installed automatically.

An overview of the data flow for a printer class device is shown in The data flow is as follows:

1. This flow is the normal Presentation Manager API to the printer.

    The application program opens a printer DC, the data is then written (eg using GPI function), and the DC is then closed.

2. The data may be passed to part I of the Presentation Manager Printer Device Driver. This would depend upon the format of the spool file to be created.

3. The data is passed to the Spooler Queue Manager, which creates a print job. There is a queue of print jobs per logical device address (eg "LPT1").

   A print job is created for each *StartDoc/EndDoc* escape sent to the printer DC.

4. This flow is the top of a queue going to its Spooler Queue Processor. A spool file is read by the processor, and is deleted by the Queue Manager when the processor has finished.

   Presentation Manager provides two processors: PRINT (for raster printer class devices) and PLOT (for vector plotter class devices). User processors can also be defined to the Spooler, and the Presentation Manager ones can be replaced. This is done via the Control Panel (see the section, "Control Panel" in Chapter 3).

5. This flow is the Spooler Queue Processor using the normal API to write to a printer.

   A Spooler Queue Processor performs whatever operations are appropriate for its queue. In the case of the Presentation Manager PLOT Processor, reverse clipping is performed if requested and the datatype is suitable, before the data is passed to the API. This is because vector plotter class Device Drivers do not do this, unlike raster printer class Device Drivers.

6. This flow is to Part II of the Spooler Printer Device Driver for the required printer.

   User Device Drivers can also be defined to the Spooler, and the Presentation Manager ones can be replaced. This is done via the Control Panel (see the section, "Control Panel" in Chapter 3).

7. This flow is to the DOS Device Driver.

8. The DOS Device Driver sends data to the printer via the hardware adapter.

   The main flow is now complete. The following covers miscellaneous flow:

9. Data can flow between a Presentation Manager Printer Device Driver and an application for various reasons, for example, the application can query a Device Driver (eg for banding, plotter pen allocation).

10. An application can write directly to a spool file, bypassing the Presentation Manager Device Driver Part I. This would be done if, for example, an application wished to write to the printer using a device dependent printer datastream.

11. This is a user interface to the Spooler queues. This enables a user to query and manipulate the queues.

    The following applies to the replacement of the DOS Spooler:

12. This flow is the normal DOS API to the printer.

    The application issues a *DosOpen* for the printer, followed by 'n' *DosWrites/DosIoctls*, followed by a *DosClose*.

    When the *DosOpen* is issued, an Open Ioctl is sent to the DOS printer Device Driver, followed by an Activate Font Ioctl.

13. This flow is from the printer Device Driver to the DOS Spooler emulation.

14. This flow is to the Queue Manager as described in A print job is created for each *DosOpen/DosClose*.

15. This flow is the Presentation Manager Printer Device Driver Part II registering and deregistering itself as writing directly to the DOS device.

16. This flow is the SPOOL command querying whether data should be spooled or written back to the DOS Device Driver.

17. This flow is the PRINT command switches causing one or more jobs on a spooler queue to be cancelled.

18. This flow is the PRINT command printing a file, and is similar to.

The Spooler includes support for print devices connected via the DOS Async Device Driver (COMn), and the Presentation Manager supplied IEEE DOS Device Driver. Note that for technical reasons, the DOS Spooler does not spool data destined for COMn devices, only LPTn devices; and since the Presentation Manager Spooler emulates the DOS Spooler, the same restriction applies. Thus if a DOS application sends data to COM1 directly, it could be interleaved with output from the Presentation Manager Spooler or another DOS application. Data from a Presentation Manager application destined for COM1 would of course be spooled.

11.1.3 Spooler and Printer Configuration

The user can tailor the Spooler using the Control Panel (see the section, "Control Panel" in Chapter 3). This section outlines the tasks to be performed to install and setup the Spooler, and connect a printer.

11.1.3.1 Spooler Installation

For the Spooler the following is specified:

- Spooler

  If *install* is specified, Presentation Manager runs with the Spooler installed.

  If *noinstall* is specified, Presentation Manager runs without the

Spooler installed. See the section, "Spooler Not Installed", for
further information on the data flow if the Spooler is not installed.

11.1.3.2 Logical Device Address

For each device queue (eg "LPT1"), the following is specified:

- Default Spooler Queue Processor Name

  This is used if a name is not supplied by the application (the initial
  default is PRINT).

- Alternative Device

  This is optional, but if specified causes all data directed to this
  device to be sent to the alternative device.

- Spool Directory

  The drive letter and name of the Spool directory (C:\SPOOL is the
  initial default).

- SPOOL Command

  This is only applicable to devices supported by the DOS SPOOL
  command (see the section, "DOS SPOOL and PRINT Commands").

  If *execute* is specified, then the user does not have to execute the
  SPOOL command, since it is done automatically using the
  appropriate options specified for this device (eg spool directory).

  If *noexecute* is specified, then execution of the SPOOL command is
  left to the user.

11.1.3.3 Queue Processors

A processor file can be added or deleted.

The following is specified:

- Any processor dependent options

11.1.3.4 Printer Device Drivers

A Presentation Manager Printer Device Driver file can be added or deleted.

The following is specified:

- Default datatype

  Datatype format of the spool file if not specified by the application.
  What datatypes are available depends upon the Device Driver (see

the section, "Spool File Data Format").

- Any Device Driver options

Note: When a Device Driver is specified for a printer (see the section, "Printers"), the Device Driver needs to obtain Device Driver/printer related data.

11.1.3.5 Printers

Printers may be added or deleted. The following is specified for each printer:

- Spooler Printer Device Driver to be used.
- Device Driver/Printer information

  This must include:

  - Forms Code

    User specified forms code (upto 24 characters), with information on the size of the form and any user/hard clip limits.

    The user must also specify which one is installed.

  - Paper Orientation,

    Whether the paper is to be treated as *landscape* or *portrait*.

  - Device Dependent Data

    This depends upon the device (eg plotter pen color allocation).

- Printer address

  This is optional, and if not specified the printer cannot be used. Addresses are LPT1-3, COM1-8, IEEE1-14, PRN=LPT1 and AUX=COM1.

  For an IEEE connected device, the physical address must also be specified for the logical address IEEEn.

- User name for printer. This is for reference by applications who may wish to present the user with a user specified name.

The following global information is also specified:

- Default Printer

  A printer may be specified as the default. This is for reference by applications to determine which printer to use, if they do not wish to present the user with a list of printers available.

11.1.4   Spool File Datatype Format

The format of a spool file is known as its datatype, and each datatype is identified by a datatype name.

Each call to a Processor will be from the main thread of a DOS process, except for the Query and Control functions. Being invoked in this manner overcomes some of the reentrancy problems for the Processor. A Processor can start other DOS threads or processes if it wishes.

11.1.4.1   Predefined Datatypes

The current predefined datatype names and formats are as follows:

- Q_STD

    The spool file created is in a format which is printer and Device Driver independent.

- Q_ESC

    The spool file created is in a format which is printer independent, but Device Driver dependent.

    This datatype is a general mechanism for Part I of a Spooler Device Driver to communicate to Part II. The actual format is identified by an Escape Number, and there is a number for a predefined format, called Journal. It is also possible to have user defined formats, with user defined Escape Numbers.

- Q_RAW

    The spool file created is in a format which is printer dependent: a printer datastream.

In general, the format defined by the datatype name Q_STD should be used, although there are times when one of the others would be more suitable. A list of points which influence the choice of datatype follows:

1. Printer and Device Driver Independence.

    The more independent a datatype is, the greater the chance of the achievement of visual fidelity if a spool file is rerouted to another type of printer. Thus with Q_STD, visual fidelity can normally be achieved. However, with Q_ESC and Q_RAW it is unlikely to occur.

2. Disk Space

    In general, Q_STD requires least disk space, and Q_RAW most. It would not be unusual for a picture which takes up 10K bytes when in Q_STD format to take up 100K when in Q_RAW format.

3. Spooler Queue Processor

   It is only practical for a Spooler Queue Processor to analyse a spool file with Q_STD datatype. In particular, the Presentation Manager supplied PLOT Queue Processor only performs reverse clipping on Q_STD.

4. Performance and Program Size

   The time taken to format a spool file is in general less for a printer dependent format then for an independent format. Thus Q_RAW is quicker to format than Q_ESC, which is slightly quicker than Q_STD.

   Similarly, the program to format datatype Q_RAW is much smaller than that required for Q_ESC, which in turn is smaller than that for Q_STD.

5. Optimised Banding

   The only datatype which is suitable for optimised banding is Q_ESC (see the section, "Printing Using Banding").

All Presentation Manager supplied Spooler Queue Processors and Printer Device Drivers support these types.

11.1.4.2   User Datatypes

User datatypes can have any name. They are not supported by the Presentation Manager supplied Spooler Queue Processors and Printer Device Drivers.

They can be generated and processed as follows:

1. A user Printer Device Driver can generate a spool file in a user format. The file must be processed by a user Queue Processor which understands the format, and possibly the user Printer Device Driver - Part II, would also need to understand the format.

2. An application can create a spool file directly in a user format. The file must be processed by a user Queue Processor which understoods the format, and possibly the User Printer Device Driver - Part II, would also need to understand the format.

11.1.5   Application Printing

11.1.5.1 How an Application Prints

An application can print using the following methods:

1. An application can write to a printer using the normal API (eg the GPI for graphics) as follows:

   1. A printer DC is created by the DevOpenDc function, specifying queued output (see the chapter, "Device Contexts").

      The Presentation Manager initialisation file can be queried to find such information as which printers are attached, the driver and device name for each printer, which Spooler Queue Processors are installed (see the section, "The Initialization File").

   2. Output is delimited by the *StartDoc* and *EndDoc* escape codes, using the *DevEscape* function. In addition, each new page is begun with a *NewFrame* escape code (*EndDoc* causes a new page). All data in a *StartDoc/EndDoc* is printed together.

   3. The data is written to the printer using the API:

      1. A PS is associated with the printer DC.

      2. Data is drawn into the PS (eg a draw chain or a play metafile could be performed for a GPI PS).

      3. The PS is disassociated from the printer DC.

   4. The Printer DC is closed by the *DevCloseDc* function.

   This method is particularly suitable for graphics.

2. An application can write to the spool queue direct (for subsequent printing) by using Spooler Queue Manager function calls as follows:

   1. A Queue Manager open is performed.

   2. Output is delimited by the Queue Manager *StartDoc* and *EndDoc* escape functions calls (*EndDoc* causes a new page).

   3. The data is written using the Queue Manager Write function call.

   4. A Spooler Queue Manager close is performed.

   This method is particularly suitable for alpha, or where an application 'knows' the printer datastream and wishes to produce the datastream itself.

3. An application can write to a printer using the normal DOS API (ie *DosOpen, DosWrite* and *DosClose*). This method is particularly suitable for alpha.

Note: It could take some time to spool or print. So for an application to be 'well-behaved', it should print on a separate thread from its input processing. See the chapter, "How to Write a Presentation Manager Application", for details on 'well behaved' applications.

11.1.5.2 Printing Using Banding

Banding is a technique in which an image or picture is printed by dividing it into several bands (also known as "swathes" or "slices") and each band is sent to the printer separately. This reduces tha amount of memory needed to hold a picture before it is sent to the printer, since it is not necessary to hold the entire picture in memory. Banding can be used with any printer that has a banding capability (eg a raster class printer).

An application does not have to do anything to ensure that banding is used, since it is automatically used if the printer has banding capability.

An application can optimise banding if it is producing a spool file of data-type Q_ESC, the format (as indicated by the Escape Number) supports it (Journal format is suitable), the Device Driver supports it and the data is not going to be formatted in some way by the Spooler Queue Processor. This optimisation is done by sending only the data applicable to a particular band (eg a title line could well apply only to the first horizontal band).

To print using optimised banding, the following should be done:

1. A printer DC is opened, with a datatype of Q_ESC.
2. Use the DevQueryDeviceCaps function to ensure that banding support is provided by the Device Driver.
3. Use the DevEscape function with the NextBand escape code to retrieve the coordinates of a band, which will be a rectangle.

   Coordinates are in device units, and all subsequent calls (eg GPI) are clipped to this rectangle.
4. If the coordinates give an empty rectangle, then the end of the banding operation has been reached. In this case, close the printer DC.
5. Use the GpiConvert function to translate from device to world units.
6. Use GPI and other functions to draw in the band.
7. Repeat steps to, as necessary.
8. Close the printer DC.

11.1.5.3 Aborting a Print Operation

The *DevEscape* function with the *AbortDoc* escape code can be used to abort a print operation (see the chapter, "Device Contexts").

11.1.5.4   Errors During Printing

If any errors occur during the actual printing process, they are normally
reported to the user (eg if a Device Driver encounters an error, such as 'out
of paper', it reports it to the user for resolving, and recovers from it if pos-
sible). However, if an error occurs when generating a spool file, it is not
reported to the user - a bad return code is returned to the application and
it is left to the application to handle it (eg it could report it to the user
and then try again if requested).

11.1.6   Device Driver - Part I

The purpose of this section of a Device Driver is to support queries from
the application, and to assist the application API in the production of a
spool file.

11.1.6.1   Application Query Support

The application can query the driver to obtain information on the capabil-
ities of the printer. This is done via the DevQueryCaps and similar func-
tion calls (see the chapter, "Device Contexts").

11.1.6.2   Spool File Creation

Logically the spool file is created by the Device Driver. However, the
amount of involvement of the Device Driver in the creation of the spool file
depends upon the datatype of the file to be produced.

For the predefined datatypes, processing is as follows:

- Q_STD

  This independent format only uses the Device Driver to handle
  Queries. Thus a Device Driver just has to be able to handle any
  Queries and return the result, to support this datatype.

  The Device Driver creates the spool file by getting Presentation
  Manager to generate it.

- Q_ESC

  The Device Driver is very much involved in this since this format is
  to be interpreted by Part II of the Device Driver: the spool file con-
  sists of essentially an Escape for Part II of the Device Driver. Thus
  the Device Driver has to be able to produce device driver dependent
  data, and handle any Escapes or Queries, to support this datatype.

  The Device Driver creates the spool file, although assistance is
  available from Presentation Manager to produce the Journal

format.

- Q_RAW

  The Device Driver is very much involved in this since the data is esssentially a printer datastream. Thus the Device Driver has to be able to produce a printer datastream for its printer, and handle any Escapes or Queries, to support this datatype.

  The Device Driver creates the spool file.

User datatypes are essentially generated by the Device Driver, so the Device Driver is fully involved.

11.1.6.3 Presentation Manager Supplied Device Drivers

Device Drivers are supplied for a wide range of printers. They only support the Presentation Manager defined datatypes.

11.1.6.4 User Device Drivers

For general use, a user Device Driver should support at least the Presentation Manager datatypes Q_STD and Q_RAW.

For information on how to produce a user Device Driver see the chapter, "Device Driver Interface".

11.1.6.5 Device Driver Functions

For a detailed description of the function calls see the chapter, "Device Driver Interface".

11.1.7 User Interface

A user interface is provide to manage the queues and the print jobs.

11.1.7.1 Function

When invoked, a list of the queues is displayed, with summary information, such as the number of print jobs in each queue. This list may be scrolled, and various operations may be performed on a queue.

The jobs in each queue are displayed in priority order (for details on when a job may be printed, see the section, "Job Selection"). The jobs in the queue may be scrolled, and various operations performed on the jobs, which may affect a job's priority. Against each job, information such as

its state (eg hold), and the relevant data passed on the Open function call (eg comment) is displayed.

11.1.7.1.1 Hold Queue

The selected queue is placed into the hold state. No jobs are printed from a held queue, any currently printing jobs are allowed to complete.

This function cannot be used if the queue is already held.

In general, a user should hold a queue for a particular printer before changing the forms, with the new installed forms being specified via the Control Panel.

11.1.7.1.2 Release Queue

The selected queue is changed from the held state into the active (or empty) state. This function can be used only if the queue is in the held state.

11.1.7.1.3 Print Job Next

This function operates on a single specified job. That job is moved from its current position in its queue to the top of the same queue, unless already printing. The job will become the next job printed when the target printer becomes free, subject to the selection rules (see the section, "Job Selection").

A job in the Held state will be released by this function.

11.1.7.1.4 Print Job Now

This function is essentially the same as Print Job Next. The only difference is that, after the specified job has been moved to the top of the its queue, any other job currently printing on the target printer is then *cancelled*

11.1.7.1.5 Cancel Job

The selected job is removed from the queue. If any of the selected jobs is being printed, then the printing operation is interrupted.

11.1.7.1.6 Restart Job

The selected job, which must be the job currently printing, is interrupted and is then restarted from the beginning.

11.1.7.1.7 Repeat Job

The selected job is duplicated and is placed onto the queue as if it had just been created as a completely new job.

If the original job is held then the repeated job will also be held.

11.1.7.1.8 Hold Job

The selected job or jobs are placed into a hold state. In this state, a job will not be selected for printing when it reaches the top of the queue. If the job is being printed, the print is paused.

This function cannot be used when a job is already in the hold state.

11.1.7.1.9 Release Job

The selected job or jobs are placed into the Ready to Print state.

The function can be used only if the selected job or jobs are in the held state.

11.1.7.1.10 Redirect Job

This function can be used to change the queue of a specified job.

If the selected job is being printed, the print is first aborted and then the job redirected.

11.1.8 Queue Manager

The purpose of the Spooler Queue Manager is to look after the queues, creating new spool files and print jobs when necessary, and invoking Spooler Queue Processors to process them.

11.1.8.1 Job Selection

A job's initial position in a queue depends upon its priority: it is put in
front of all jobs with a lower priority, but after all jobs with an equal or
greater priority. This priority may change in response to requests issued
by the user (eg Print Job Next). When a job gets printed depends upon
the selection algorithm, which is outlined below:

1. Find the jobs with the highest priority in the queue for the device,
   which have the correct forms code (ie the forms installed in the
   printer (see the section, "Printers") matches that required by the
   job, or the job has no forms code) and are not held.

2. If there is more than one such job, choose the one which is nearest
   to the head of the queue.

11.1.8.2 Print Job Generation Functions

SplQmOpen

```
HSPL SplQmOpen(token, length, data)
LPSZ token;
LONG length;
LPBUF data;
```

This function corresponds to the *DevOpenDC* function: it
opens the Spooler Queue Manager for generating a print job.

Parameters:

---

token A string containing a token (nickname) which
identifies Spooler information held in the
PRESSERV.INI file. This information is the same
as that which may be pointed to by *data*; any that
is obtained in this way overrides the information
obtained using *token*.

   If *token* is specified as *"", then no device informa-
tion is taken from PRESSERV.INI. Presentation
Manager Release 1 requires *"" to be specified.

length The length of *data* supplied. This may be shorter
than the full list if omitted items are irrelevant or
supplied from *token* or elsewhere.

data A parameter block containing:

```
struct DOPDATA
     LPSZ driver_name;
     LPBUF driver_data;
     LPSZ log_addr;
     LPSZ data_type;
```

```
LPSZ comment;
LPSZ proc_name;
LPSZ proc_params;
LPSZ spl_params;
LPSZ network_params;
 ;
```

driver_ name
> A string containing the name of the Device Driver (eg "EPSON"). This information must always be supplied if it is not available from *token*.

driver_ data
> Data which is to be passed directly to the Device Driver. Whether or not any of this is required depends upon the Device Driver, though the information may alternatively have been specified via *DevSetEnvironment*.

> The data consists of the following:

```
struct DRIVDATA
     LONG length;
     LONG version;
     SZ device_name;
     LPBUF general_data;
      ;
```

> length   The length of the whole *driver_ data* structure.

> version   The version number of the data. Version numbers are defined by particular Device Drivers.

> device_ name
> > A string in a 32-byte field, identifying the particular device (model number etc). Again, valid values are defined by Device Drivers.

> general_ data
> > Data as defined by the Device Driver.

log_ addr
> The logical address of the output device (eg "LPT1"). This is optional, since the Spooler will provide a default if

necessary.

data_type

This defines the type of data which is to be queued, as follows:

- Q_STD - standard format
- Q_ESC - escape format
- Q_RAW - raw format

Note that a Device Driver may define other datatypes. This information must be supplied if it is not available from *token.*

comment

A natural language description of the file. This may, for example, be displayed by the Spooler to the end user. It is optional.

proc_name

The name of the queue processor. This will normally be defaulted.

proc_params

A parameter string for the queue processor, and is optional.

spl_params

A parameter string for the Spooler, which is optional. This has the following options, which must be separated by one or more blanks:

- FORM=f

  Specifies a forms code 'f'. This must be a valid forms code for the printer (see the section, "Printers").

  If not specified, then the data is printed on the forms in use when this print job is ready to be printed.

- PRTY=n

  Specifies a priority in the range 0-99, with 99 being the highest. If not specified, then a priority of 50 is used.

network_params

A parameter string for networking, which is optional. In general an

application would leave it to the Network Program to specify this parameter, since it is only applicable if the logical address has been rerouted.

This data will be ignored in Presentation Manager Version 1.

Returns:

```
0 Error
!=0 Spooler handle
```

SplQmStartDoc

```
BOOL SplQmStartDoc(hspl, docname)
HSPL hspl;
LPSZ docname;
```

This function corresponds to the *StartDoc* escape function: it specifies that a new print job is starting.

Parameters:

hspl   Spooler handle.

docname
    Document name. This may be displayed by the Spooler to the end user.

Returns:

```
0 Error
1 OK
```

SplQmWrite

```
BOOL SplQmWrite(hspl, length, data)
HSPL hspl;
LONG length;
LPBUF data;
```

Write a buffer to the spool file for the print job.

Parameters:

hspl   Spooler handle.

length   Length of *data*.

data   Buffer of data to write to the spool file.

Returns:

```
0 Error
1 OK
```

SplQmEndDoc

```
LONG SplQmEndDoc(hspl)
HSPL hspl;
```

This function corresponds to the *StartDoc* escape function: it ends a print job, and returns its id, which is a unique number to identify the job.

Parameters:

hspl   Spooler handle.

```
0 Error
!=0 Jobid (1-65535)
```

SplQmClose

```
BOOL SplQmClose(hspl)
HSPL hspl;
```

This function corresponds to the *DevClose* function: it closes the Spooler Queue Manager.

Parameters:

hspl   Spooler handle.

Returns:

```
0 Error
1 OK
```

SplQmAbort

```
BOOL SplQmAbort(hspl)
HSPL hspl;
```

This function aborts the generation of the spool file(s). It automatically closes the Spooler Queue Manager (see *SplQmClose*).

Parameters:

hspl   Spooler handle.

Returns:

```
0 Error
1 OK
```

Note: If *SplQmStartDoc* or *SplQmEndDoc* are not specified correctly (eg one is missing), then sensible defaults are assumed.

11.1.9 Queue Processor

A Spooler Queue Processor takes a spool file, and prints it. There is a processor for each queue, with two being supplied with Presentation Manager: PRINT and PLOT. In addition, user processors may be used.

11.1.9.1 How a Queue Processor Prints

A processor is called by the Spooler Queue Manager, and is passed such information as the processor parameters, plus the data to print.

How a processor writes to a printer depends upon the datatype as follows:

1. Q_STD

   This data is written using the API in a manner similar to that used by an application (see the section, "Application Printing"):

   1. A printer DC is created by the DevOpenDc function (see the chapter, "Device Contexts"), with a request for direct output.

   2. Output is delimited by the *StartDoc* and *EndDoc* escape codes, using the *DevEscape* call.

   3. The data is written to the printer using the Presentation Manager API. The processor can perform any operation it wishes on the data before sending it to the API (eg the Presentation Manager Plot Class Processor performs reverse clipping if requested). In particular, the Presentation Manager GPI function are available, including metafile support (eg a GPI PS is associated with the DC, and data is drawn to the PS using GPI and/or metafile functions).

      The data is in metafile format. Thus MetPlayMetaFile (see the chapter, "Metafile Support") could be used to 'print' the data. Alternatively, the metafile could be analysed and 'printed' using the GPI.

   4. The Printer DC is closed by the *DevCloseDc* call.

2. Q_ESC

   This data is written using the DevEscape function call:

   1. A printer DC is created by the DevOpenDc function (see the chapter, "Device Contexts"), with a request for direct output.

   2. Output is framed by the *StartDoc* and *EndDoc* escape codes, using the *DevEscape* call.

   3. The data is written to the printer using the *DevEscape* function call. The data is broken into blocks of a convient size to be handled by the processor, and each block is written using the Escape number specified for the spool file.

4. The Printer DC is closed by the *DevCloseDc* call.

3. Q_RAW

This data is written using the *DevEscape* function call:

1. A printer DC is created by the DevOpenDc function (see the chapter, "Device Contexts"), with a request for direct output.

2. Output is framed by the *StartDoc* and *EndDoc* escape codes, using the *DevEscape* call.

3. The data is written to the printer using the *DevEscape* function call. The data is broken into blocks of a convient size to be handled by the processor, and each block is written using the RawData option.

4. The Printer DC is closed by the *DevCloseDc* call.

4. User Datatype

This is very much datatype dependent. The processor can perform any type of processing it wishes to on the data before outputing it. It can send the data to the printer using any of the methods described for the datatypes. It could even requeue it, or throw it away.

If required, a Queue Processor can issue messages via the the section, "Message Interface".

If a Queue Processor needs to create temporary files on disk, it should in general create a temporary subdirectory off the spool directory for the device address, and create the files in there.

11.1.9.2 Queue Processor Functions

The interface provided by a Spooler Queue Processor is as follows:

SplQpOpen

```
HPROC SplQpOpen(length, data)
LONG length;
LPBUF data;
```

Open the processor.

Parameters:

---

length   The length of *data* supplied. This may be shorter than the full list if omitted items are irrelevant.

data   A long pointer to a parameter block containing:

```
struct QPDATA
```

```
        LPSZ driver_name;
        LPBUF driver_data;
        LPSZ log_addr;
        LPSZ data_type;
        LPSZ comment;
        LPSZ proc_params;
      ;
```

driver_ name

A string containing the name of the Device Driver (eg "EPSON").

driver_ data

Data which is to be passed directly to the Device Driver. Whether or not any of this is required depends upon the Device Driver, though the information may alternatively have been specified via *DevSetEnvironment*.

The data consists of the following:

```
struct DRIVDATA
    LONG length;
    LONG version;
    SZ device_name;
    LPBUF general_data;
    ;
```

length   The length of the whole *driver_ data* structure.

version   The version number of the data. Version numbers are defined by particular Device Drivers.

device_ name (SZ)

A string in a 32-byte field, identifying the particular device (model number etc). Again, valid values are defined by Device Drivers.

general_ data

Data as defined by the Device Driver.

log_ addr

The logical address of the output device (eg "LPT1").

data_ type
> This defines the type of data which is to be processed, as follows:

- Q_STD - standard format

- Q_ESC - escape format

- Q_RAW - raw format

> Note that a Device Driver may define other datatypes.

comment
> A natural language description of the file. This may, for example, be displayed by the Spooler to the end user. It is optional.

proc_ params
> A parameter string for the queue processor, and is optional.

Returns:

> *0* Error
> *!=0* Queue Processor handle

SplQpPrint

```
BOOL SplQpPrint(hproc, filename)
HPROC hproc;
LPSZ filename;
LONG lastfile;
```

Process and print the spool file. The Processor assumes that printing starts on a new page, and issues a form feed after printing a file.

Parameters:

hproc   The processor handle.

filename
> Name of file containing data to be processed.

Returns:

> *0* Error
> *1* OK

SplQpClose

```
BOOL SplQpClose(hproc)
HPROC hproc;
```

Close the processor.

Parameters:

hproc   The processor handle.

Returns:

0 Error
1 OK

SplQpControl

```
LONG SplQpControl(hproc, code)
HPROC hproc;
LONG code;
```

This function controls the Queue Processor.

Parameters:

hproc   The processor handle.

code   The control code.

> **1 - ABORT**
> The print is aborted. The Queue Processor is automatically closed.
>
> **2 - PAUSE**
> The print is paused.
>
> **3 - CONTINUE**
> A paused print is continued.

Returns:

0 Error
1 OK

SplQpQueryDt

```
BOOL SplQpQueryDt(length, count, array)
LONG length;
LONG *count;
LPSZ array[*count];
```

This function returns a list of the datatypes supported.

Parameters:

length   The maximum length of a datatype name.

*count   The maximum number of datatypes that can be returned. On return, this is updated to the number actually returned.

array[*count]
             An array, which on return contains the datatypes supported. The length of each array element is *length,* and each ASCIIZ datatype name is

truncated to length *length,* if necessary.

Returns:

0 Error
1 OK

SplQpInstall

```
BOOL SplQpInstall(hwnd)
HWND hwnd;
```

This function performs any installation work that is required. It is called by the Control Panel (see the section, "Control Panel" in Chapter 3) when the Queue Processor is installed. The processor then holds a conversation with the user to obtain any processor dependent options. These are then stored in the PRESSERV.INI file.

Parameters:

hwnd (HWND)
Window handle.

Returns:

0 Error
1 OK

11.1.9.3   Presentation Manager Supplied Queue Processors

Two Queue Processors are supplied (PRINT and PLOT), which support the predefined datatypes (Q_STD, Q_ESC, Q_RAW).

11.1.9.3.1   PRINT

This processor is aimed at raster printers, although it could be used with vector plotters. PLOT (see the section, "PLOT") should be used with plotters if plotting function such as reverse clipping is required. It sends data to the printer without performing any operations on it, or making any changes. *Parameter*

The ASCIIZ processor parameter string takes the following options:

1. COP=n

    Print 'n' copies (n is an integer greater than zero).

    Default is 1. *Note:* The forms used in the printer must be the same for all copies: the forms in the printer for the first page of the first copy must not be changed until the last page of the last copy has been printed. If this is not followed, the output may be incorrect.

2. The following options only apply if the datatype is Q_STD:

  1. COL=M/C.

     Produce mono output (M) (no background, black foreground), or color (C) output.

     Default is M.

  2. MAP=N/A

     The colors are printed in a normal way, where background default is white and neutral white is black (N); or asis, where background default is black and neutral white is white (A).

     Default is N. *Note:* If color is requested on a mono printer, the Device Driver will, as expected, only produce mono output. Which means that a red line on a blue background would not be seen, since both would have been printed as black. Thus, the mono option does in general produce a better result in this sort of configuration, since any background is removed.

  3. ARE=C/w,d,l,t

     The output area size is the area within the clip limits (C), or is given as the width (w), depth (d), offset from the left (l), and offset from the top (t). The units are in terms of a percentage of the size of the paper.

     Default is C.

  4. ORI=L/P

     The orientation is landscape (L) or portrait (P).

     Default is portrait.

  5. FIT=S/l,t

     The output is fitted to the output area by scaling to fit (S) (ie the output is scaled until the larger of height or width just fits within the output area), or is fitted as real size, with the centre of the output area mapping on to a point in the output which is offset from the left (l) and offset from the top (t) of the total output. The units are in terms of a percentage of the size of the output.

     Thus if real size is required and the output area is one quarter of the total output, then printing in the bottom quarter could be obtained by specifying l and t as 75.

     Default is S.

The options must be separated by one or more blanks as in the following example of the parameter string:

```
COP=2 FIT=S@
```

where @ is hex zero.

Any errors detected in the parameter are ignored, and the default value for an option is used if an option is not supplied or is in error. *Restrictions*

The following restrictions apply when the datatype is Q_STD:

1. Any GPI restrictions, such as scaling of image (see the chapter, "Graphics Programming Interface").

2. Any metafile restrictions (see the chapter, "Metafile Support").

11.1.9.3.2 PLOT

This processor is aimed at vector plotters, although it could be used with raster printers. The difference between this processor and PRINT (see the section, "PRINT") is that this processor can perform reverse clipping on spool files of datatype Q_STD (this is automatic with raster printers), and runs slower because of the general extra processing involved. Reverse clipping is not normally performed by Presentation Manager vector plotter Device Drivers because of the complexity of the operation.

This processor also performs a color sort so that if the device driver does not support a pen sort, the user is not continually being told to change pens. *Parameter*

The ASCIIZ processor parameter string takes the following options:

1. COP=n

    See the section, "PRINT".

2. The following options only apply if the datatype is Q_STD:

    1. MOD=D/H/F.

        The plotter mode is draft, where hidden lines are shown and fill patterns and shading are not plotted (D); or hidden, where areas are filled but hidden lines are still plotted (H); or full, where full reverse clipping is performed (F).

        Default is D.

    2. CHA=N/D/A

        The plotter characters used are none, where the typefaces defined in the output are used (N); or default, where the plotter character set is used to plot characters that use the default typeface (D); or all, where the plotter character set is used to plot all characters (A).

        Default is A.

3. ARE=C/w,d,l,t

   See the section, "PRINT".

4. ORI=L/P

   See the section, "PRINT".

5. FIT=S/l,t

   See the section, "PRINT".

The options must be separated by one or more blanks as in the following example of the parameter string:

COP=2 FIT=S@

where @ is hex zero.

Any errors detected in the parameter are ignored, and the default value for an option is used if an option is not supplied or is in error. *Restrictions*

The following is not supported when the datatype is Q_STD:

1. Image.

2. Any restrictions imposed by the conversion program PicIchg (see the chapter, "Printing Interface") when converting from metafile format to PIF.

11.1.9.4 User Queue Processors

For general use, a user queue processor should support the predefined datatypes (Q_STD, Q_ESC, Q_RAW). For further details see the section, "How a Queue Processor Prints".

11.1.10 Device Driver - Part II

The purpose of this section of a Device Driver is to output to the printer, via the appropriate DOS Device Driver.

A Device Driver can direct the user when necessary. For example, if output is to a plotter device, the application does not have to concern itself with the color mapping of pens: the application uses the normal GPI say, and specifies color using GPI functions and a color table. The Device Driver then maps these colors to the physical pens, which may mean that more than one carousel is needed (this mapping is performed when the plotter is installed). In this case, the Device Driver puts up a message directing the user to perform the change using the "Message Interface".

11.1.10.1 Presentation Manager Supplied Device Drivers

Device Drivers are supplied for a wide range of printers.

11.1.10.2 User Device Drivers

For information on how to produce a user Device Driver see the chapter, "Device Driver Interface".

11.1.10.3 Device Driver Functions

For a detailed description of the function calls see the chapter, "Device Driver Interface".

11.1.11 DOS Device Drivers

The DOS Printer and Async Device Drivers are used unchanged. In addition, an IEEE Device Driver is supplied with Presentation Manager, and it is used with IEEE plotters.

11.1.12 Message Interface

The Spooler has a Message Interface to centralise the displaying of messages.

This must be used by any part of the Spooler that wishes to output a message (eg Queue Processors, Device Drivers). For example, by Device Drivers to display user prompts, such as change plotter pens.

11.1.13 Message Functions

SplMessageBox

```
UINT SplMessageBox(log_addr, err_info, err_data,
                   text, caption, style)
LPSZ log_addr;
UINT err_info;
UINT err_data;
LPSZ text;
LPSZ caption;
UINT style;
```

This function creates and displays a message box, and waits for the user response. The parameters are based upon those of WinMessageBox (see the section, "Message Boxes"), a fuller description.

Parameters:

log_ addr

>The logical address of the output device (eg "LPT1").

err_ info

>Error information, defined as follows:

>One of the following bits must be set:

>*Bit 0* - Spooler Queue Processor error
>*Bit 1* - Presentation Manager Device Driver error
>*Bit 7* - Other

>One of the following bits must be set:

>*Bit 8* - Information
>*Bit 9* - Warning
>*Bit 10* - Error (recoverable error)
>*Bit 11* - Severe (unrecoverable error)

err_ data

>Error data, defined as follows:

>One of the following bits must be set:

>*Bit 0* - Printer jam (eg offline, no power)
>*Bit 1* - Form change required
>*Bit 2* - Cartridge change required
>*Bit 3* - Pen change required
>*Bit 4* - Data error (eg file missing)
>*Bit 5* - Unexpected DOS error
>*Bit 15* - Other

text   A string containing the text.

caption   A string containing the caption.

style   A bit array specifying the contents and function of the message box.

Returns:

A value which indicates the user's response.

11.1.14   DOS SPOOL and PRINT Commands

The Presentation Manager Spooler provides SPOOL and PRINT commands compatible with the MS OS/2 Spooler.

The Spooler switches on the PRINT command operate on the Presentation Manager spooler queue for the device specified (or LPT1 if no device was specified).

If spooling of data generated by the PRINT command or by applications which write directly to a DOS Device Driver is required, then the SPOOL command should be invoked. Such data may be interleaved with other such data and with data from the Presentation Manager spooler if SPOOL is not invoked.

It is possible to setup the Presentation Manager Spooler so the the DOS SPOOL command is automatically executed. See the section, "Spooler and Printer Configuration".

11.1.14.1 Device Driver Register

The purpose of the Device Driver register component is to track which Presentation Manager Device Driver part II (if any) is writing directly to each DOS device. Each Device Driver part II (DD-II) should register itself before opening a device and should deregister itself after the device has been closed.

The SPOOL command will detect data which is being written directly by a DD-II to a device against which the DD-II is registered and will write this data to the DOS device rather than spooling it.

The registration is by process. Each device has at most one process which is registered against it at any instant.

11.1.14.2 Device Driver Register Functions

SplDdrReg

```
BOOL SplDdrReg(dev)
LPSZ dev;
```

This function registers the invoking process against the device. Any data which is subsequently written to the device by that process will not be spooled.

Parameters:

dev   Device name (eg "LPT1")

Returns:

```
0 Error
1 OK
```

SplDdrDeReg

```
BOOL SplDdrDeReg(dev)
LPSZ dev;
```

This function deregisters the invoking process against the

device. Any data which is subsequently written to the device by that process will be spooled if the SPOOL command is active for that device.

Parameters:

dev   Device name (eg "LPT1")

Returns:

```
0 Error
1 OK
```

SplDdrQuery

```
LONG SplDdrQuery(dev, pid)
LPSZ dev;
UINT pid;
```

This function queries if a specified process is registered against a specified device.

Parameters:

dev   Device name (eg "LPT1").

pid   Process id.

Returns:

```
0 Error
1 Registered
2 Not registered
```

11.1.14.3 DOS Monitor Chain

The Presentation Manager Spooler SPOOL command uses a DOS Printer Device Driver monitor to intercept print data from DOS applications. Any monitors in the chain before the Spooler will see the data twice: once when it first arrives and once when it is written by the Presentation Manager Device Driver. Any monitors after the Spooler will see the data once: when it is written by the Presentation Manager Device Driver.

Any monitor before the Spooler should do one of the following:

1. Be prepared to see the data twice.

2. Use Device Driver Register (see the section, "Device Driver Register") to pick out the second time (eg it could ignore the data the second time).

3. Install itself after the Spooler, so it only sees the data once.

11.1.15   Spooler Not Installed

If the Spooler is not installed, the API described in the section, "Print Job Generation Functions", is not available, and gives a bad return code. So an application cannot create a Presentation Manager spool file, either directly or indirectly via a Printer DC.

If an application opens a printer DC, it is treated as a direct output request instead of queued. The application is logically treated as if it is a Spooler Queue Processor (see), and its API calls flow to Part II of the Spooler Printer Device Driver, which in turn sends data to the DOS Device Driver.

Chapter 12
Printing Interface

12.1 Picture Interchange, Printing and Support

12.1.1 Introduction

This section describes the various operations (such as printing), that can be performed on pictures via the Presentation Manager User Interface Shell and utilities.

12.1.2 Conventions

12.1.2.1 Print, Printing and Printers

The generic term *print* is used to describe both printing on a printer device and plotting on a plotter device. Similarly, *printer* refers to either a printer device or a plotter device. This convention is used by the Presentation Manager Spooler (see the chapter, "Spooler Interface").

12.1.2.2 Initial Values for Panels

Most of the utilities have input panels of various types (eg menu). Each time a panel is displayed, it is initialised to the settings it had last time, with initial defaults for the first time through. The settings are remembered across IPLs.

Any exceptions to this are given in the panel description.

12.1.3 Picture File Formats

A picture can be defined as a sequence of functions which enable it to be redrawn, and such functions can be stored in a file (a picture file).

A metafile is one type of picture file that can define a picture. Such a file can be exchanged between Presentation Manager applications. An API is available to create and manipulate metafiles (see the chapter, "Metafile Support").

In addition to the metafile format, there are other formats which can define a picture. In particular, the interchange format **Picture Interchange Format (PIF)**, which is used to interchange with non-Presentation Manager applications (see the sections, "Picture Interchange" and "Interchange").

12.1.4 Picture Interchange

Metafiles can be exchanged between Presentation Manager applications. However, metafiles cannot be used for interchange with non-Presentation Manager products and applications: metafile is an *exchange* format, not an *interchange* format. However, if such a product wishes to interchange pictures, it would support an interchange format.

Presentation Manager provides a utility PICICHG to convert a metafile into an interchange file and vice-versa.

This utility can be invoked from the Presentation Manager User Interface Shell, as a line command, and by an application using the API.

12.1.4.1 User Interface

PICICHG is invoked from *Start-A-Program.*

Processing is as follows:

1. A menu bar is displayed with the following options:
   - File
   - Exit

   These options are selected as necessary.

2. The usual select/action, escape and help options are available as appropriate for a menu or panel type, unless stated otherwise.

12.1.4.1.1 File

Selecting this gives a pull down menu with the following options:

- Select Files
- Convert Files

Select Files

This allows the file or files to be selected for conversion.

A scrollable list of files for the default drive and current directory is displayed. Any number of files can be selected from those listed. It is also possible to change directories, although files can only be selected from one directory at a time: changing directories loses any selected files.

The type of file (ie metafile, PIF) is derived from the filename extension, provided it is a valid default (see the section, "Naming Conventions and Other Standards"). Otherwise it is assummed to be a metafile. *Convert Files*

This allows the conversion to be performed: a metafile is converted to a PIF , and vice versa.

The destination directory is that of the source, the filename of the destination file is that of its source, with the filename extension being that of the the destination type (eg .PIF).

In addition, any resources definitions (eg symbol sets) which are imbedded in the PIF file are made external. The filename used is based on that of the destination filename (two digits are added to the end of the name, with the name being truncated if necessary), with the appropriate filename extension (see the section, "Naming Conventions and Other Standards"), and with the destination directory is the appropriate one for the resource.

12.1.4.1.2   Exit

This causes the utility to terminate.

12.1.4.2   Line Command

The format of the line command is:

PICICHG ((d:) (path) filename (.ext))

where:

    1.   The file specified is the source file.

Names of files, etc are as for the section, "User Interface".

If no parameters are specified, the User Interface is entered (see the section, "User Interface").

12.1.4.3   Application Functions

The API provided is as follows:

PICICHG

```
BOOL PICICHG (filename1, filename2, type)
LPSZ filename1;
```

```
LPSZ filename2;
LONG type;
```

This converts a metafile into an interchange file (and vice-versa), according to the type of conversion required.

Parameters:

filename1
> An ASCIIZ string giving the name of the source file.

filename2
> An ASCIIZ string giving the name of the destination file.

type
> Specifies the type of conversion:
> 1. Metafile to PIF
> 2. PIF to metafile

Returns:

0 Error
1 OK

12.1.4.4 Function Summary

12.1.4.4.1 Metafile to PIF

Note: This utility only supports metafiles which have met the interchange conversion restrictions (see the section, "Presentation Manager Restrictions" in the chapter "Metafile Support").

12.1.4.4.2 PIF to Metafile

Full PIF is supported.

12.1.4.5 Conversion Restrictions

This section describes restrictions which may affect visual fidelity. For full details of the orders, and other information, see the chapter, "Metafile Support".

An application that produces metafiles which are usually going to be converted to another format, should restrict itself to the functions and orders that can be converted to that format.

12.1.4.5.1 Metafile to PIF

1. Fonts

   It may be possible to use the default character set with a character string, otherwise not supported.

2. Regions

   Not supported.

3. Bitmaps

   Not supported.

4. Flood Fill

   Not supported.

5. Clip Area

   Not supported.

6. Color Table

   Not supported.

7. Minor restrictions which still have to be defined.

12.1.4.5.2 PIF to Metafile

None.

12.1.5 Displaying Pictures

Presentation Manager provides the utility PICSHOW to display a picture file.

This utility can be invoked from the Presentation Manager User Interface Shell, and as a line command. No direct API is provided, since an application can use the metafile replay functions (see the chapter, "Metafile Support") to display a picture.

12.1.5.1 User Interface

PICSHOW is invoked from *Start-A-Program.*

Processing is as follows:

1. A menu bar is displayed with the following options:

- File

- Exit

These options are selected as necessary.

2. The usual select/action, escape and help options are available as appropriate for a menu or panel type, unless stated otherwise.

12.1.5.1.1 File

Selecting this gives a pull down menu with the following options:

- Select Files
- Show Files

Select Files

This allows the file or files to be selected for displaying.

A scrollable list of files for the default drive and current directory is displayed. Any number of files can be selected from those listed. It is also possible to change directories, although files can only be selected from one directory at a time: changing directories loses any selected files.

The type of file (ie metafile, PIF, piclist) is derived from the filename extension, provided it is a valid default (see the section, "Naming Conventions and Other Standards"). Otherwise it is assummed to be a metafile.

Note: If the file is a PIF file, it is converted to a metafile before being displayed. *Show Files*

This allows the files to be displayed.

The type of display is prompted for, which can be *Automatic* or *Interactive*

For interactive, some form of input (eg a keystroke) is required before the next picture is displayed.

For automatic, the next picture is displayed after a time delay. The time, in seconds, is set before processing starts

12.1.5.1.2 Exit

This causes the utility to terminate.

The initial default is a 10 second time delay for automatic processing.

12.1.5.2 Line Command

The format of the line command is:

`PICSHOW ((d:) (path) filename (.ext) (options))`

where:

1. The file specified is the source file.
2. options can be one or more of the following:
   - /PIF, /MET or /PCL
   - /An

     This specifies automatic processing, with n seconds interval (10 seconds is the default). If this is not specified, interactive processing is used.

If no parameters are specified, the User Interface is entered (see the section, "User Interface").

12.1.6 Printing Pictures

Presentation Manager provides the utility PicPrint to print a picture file.

This utility can be invoked from the Presentation Manager User Interface Shell, as a line command, and by an application using the API.

12.1.6.1 User Interface

PicPrint is invoked either from *Start-A-Program*, or in response to the *Print* option being selected from the *File Cabinet*.

Processing is as follows:

1. A menu bar is displayed with the following options:
   - File
   - Control
   - Exit

   These options are selected as necessary.

2. The usual select/action, escape and help options are available as appropriate for a menu or panel type, unless stated otherwise.

12.1.6.1.1 *File*

Selecting this gives a pull down menu with the following options:

- Select Files
- Print
- Change Printer

Select Files

This allows the file or files to be selected for printing.

A scrollable list of files for the default drive and current directory is displayed. Any number of files can be selected from those listed. It is also possible to change directories, although files can only be selected from one directory at a time: changing directories loses any selected files.

The type of file (ie metafile, PIF) is derived from the filename extension, provided it is a valid default (see the section, "Naming Conventions and Other Standards"). Otherwise it is assummed to be a metafile. *Note:* If the file is a PIF file, it is converted to a metafile before being printed.
Print

This allows the files to be printed.

The number of copies is prompted for, the default is 1. *Change Printer*

This allows the printer to be changed.

A list of available printers is displayed (user name, see the section, "Spooler and Printer Configuration" in Chapter 11). Any printer from the scrollable list can be selected. The default is the current default printer as specified by the Control Panel.

12.1.6.1.2 *Control*

Selecting this gives a pull down menu with the following options:

- Options
- Paper

This option provides User Feedback.

- Area

  This option provides User Feedback.

- Picture

  This option provides User Feedback.

When printing a picture, it is not necessary to use the whole area of the paper. In particular, a printer class device cannot actually print to the left and right edges of continuous fanfold paper: there is a border. Similarly, a plotter class device cannot actually plot to the edge of the paper: there is always a border at the edge of the paper.

The inside edges of the border are know as the *Clip Limits*. The size of this border depends upon the type of device.

Some of the options result in a menu being displayed in the top right corner, with User Feedback being provided by the Paper Representation, which is displayed in the left half (which shows the picture as it will appear on the paper), and the Output Values, which is displayed in the bottom right (which show the size and position of the Output Area). shows an example of this.

For the Paper Representation, a rectangle representing the paper is shown, with a shaded area depicting the border for the current device and paper size. The output area is shown as a rectangle within the paper. If a list of source pictures has been selected, the picture corresponding to the first file in the list may be optionally displayed on the paper. If there is no such picture, it is harder to use some of the options, since there is no feedback of the affect on an actual picture. The picture should only be taken as a guide to what will be printed, it will not necessarily match the printed picture.

For the Output Values, a set of integers is displayed which are defined as a percentage of the size of the paper. Values are given for width, depth, offset from left, and offset from top. Width and depth lie in the range 1% to 100%, and the offsets lie in the range 0% to 100%. In addition, the sum of width and offset from left is less than or equal to 100%, as is the sum of depth and offset from top.

The Paper Representation and the Output Values are updated to reflect the option that has been selected, and they can also both be updated directly for certain options. *Options*

This allows device dependent options to be defined.

If the destination class is printer, then the following groups of options appear:

1. Color Use

    - Mono

      The picture is printed in black and white. Any background is not printed, and all the nonwhite colors are printed black.

    - Color

      The picture is printed so that the colors of the printed picture are as close as possible to the definition in the file. It may also be necessary to specify the *asis* (see) option to get accurate results. The result depends upon how many colors are available on the printer being used.

2. Color Mapping

    - Asis

      The colors in the picture are printed so that any background default is black, and any neutral white objects remain white.

    - Normal

      The colors are printed in the normal way: any background default is white, and neutral white objects become black.

If the destination class is plotter, then the following groups of options appear:

1. Plot Mode

    This allows selection of the mode of plotting to be used:

    - Draft Plot

      Hidden lines are shown and fill patterns and shading are not plotted.

    - No Reverse Clipping

      Areas are filled but hidden line are still plotted.

    - Full Plot

    The fuller the plot, the longer it takes to format the picture for plotting.

2. Character Definitions

    This defines the use of plotter characters:

    - All From Plotter

      The plotter uses its own character set to plot all text characters in your picture, whatever the typefaces defined in the picture.

- Default from plotter

  The plotter uses its own character set to plot only characters that use the default typeface. Other typefaces will be plotted normally.

- None From Plotter

  The typefaces defined in the picture will be used for the plot.

  If the picture does contain characters in this typeface, plotting will be quicker. However the characters in the finished picture will look a little different from those in the original picture.

Using plotter typefaces reduces the time it takes to format the picture for plotting.

Paper

This allows the paper (ie forms code) to be defined (see).

There is a scrollable list of permissible forms codes for the device to choose from. The default is the current installed one, as defined by the Control Panel. If a different forms code is selected (eg cursor moved over a forms code and button clicked), the User Feedback is updated to reflect it. The *Select* option terminates the selection, with the selected forms code being the new forms code; and the *Cancel* option terminates the selection, with the selected forms code being the one applicable when this option was entered.

Changing the paper size could affect the Output Area. If the paper size is changed, and the Output Area still fits within it, the Output Area is centered within the paper. If the Output Area no longer fits within the paper, it is reset to its clip limits for the paper.

```
+---+--------------------+--------+-----------------------+---+---^
¦ S ¦                    ¦PicPrint¦                       ¦ I ¦ U ¦
+---'--------------------'--------'-----------------------'---'---¦
¦ File    Control   eXit                        ¦  F1=Help       ¦
+--------------------------------------------------'-------------¦
¦+----------------------^  +----------+-----------------------^¦
¦¦                      ¦  ¦          ¦Paper Select¦           ¦¦
¦¦      o               ¦  +----------'------------'-----------¦¦
¦¦    -¦-               ¦  ¦  Select required paper size:      ¦¦
¦¦    / \               ¦  ¦  +-----------------+-^            ¦¦
¦¦                      ¦  ¦  ¦A4           ¦A¦ +---------^¦  ¦¦
¦¦                      ¦  ¦  ¦A5           +-¦ ¦  Select ¦¦  ¦¦
¦¦                      ¦  ¦  ¦Headed A4    ¦*¦ %---------+¦  ¦¦
¦¦                      ¦  ¦  ¦             +-¦ +---------^¦  ¦¦
¦¦                      ¦  ¦  ¦             ¦V¦ ¦Esc=Cancel¦¦  ¦¦
¦¦                      ¦  ¦  %-------------'-+ %---------+¦  ¦¦
¦¦                      ¦  ¦                    +---------^¦  ¦¦
¦¦                      ¦  ¦                    ¦ F1=Help ¦¦  ¦¦
¦¦                      ¦  ¦                    %---------+¦  ¦¦
¦%----------------------+  %-----------------------------+¦  ¦¦
¦                          +-----------------------------^¦
```

```
|                                      |  Left Margin  O       Top Margin  O   | |
|                                      |  Width       50       Depth       50  | |
|                                      %---------------------------------------+ |
%--------------------------------------------------------------------------------+
```

Figure 12.1 PicPrint Paper Panel

The final layout may not be identical to this figure. *Area*

This allows the Output Area to be manipulated, which clips the picture.

There are two groups of options to choose from:

1. Position Area

    This has the following options:

    - Rotate Area

        This is a toggle. The plotting area is repositioned by being rotated through 90 degrees: from portrait to landscape in a clockwise direction, and from landscape to portrait in an anti-clockwise direction.

        If the action is not possible, because the area would not fit in its new position, a message is displayed.

    - Centre Area

        This centres the Output Area on the paper.

2. Clip Area

    This has the following options:

    - User Values

        If this is selected, the currently defined Output Values (which show size and position) displayed are used. They can be updated via the keyboard.

    - Move Area

        This option allows movement of the Output Area by positioning the cursor (via the pointer device) on the area rectangle, and dragging the box to its new position.

        Only applicable if the area is smaller than the paper.

    - Rubber-band

        This area allows rubber-banding of the Output Area: using the cursor (via the pointer device), an edge can be moved in one direction, thus making the area bigger or smaller. The area cannot be made bigger than the paper.

- Scale

  Selecting this allows scaling of the Output Area, using the cursor (via the mouse). The aspect ratio relative to the paper is kept. The area cannot be made bigger than the paper.

- Scale to Paper

  This makes the Output Area fill the paper. The selected area is centred within the paper.

- Scale to Device

  This defines an Output Area that fills the paper excluding the border at the edge of the paper.

Note: There is no *cancel* option (a *return* option is used to return), and so any operation remains in affect, unless undone by another operation.
Picture

This allows the picture repesentation to be manipulated.

There are two groups of options to choose from:

1. Picture Control

   This has the following options:

   - Display

     This is a toggle, and this option controls whether the picture should be drawn or not.

   - Rotate

     This is a toggle, and this option rotates the representation of the paper on the screen by 90 degrees: from portrait to landscape in a clockwise direction, and from landscape to portrait in an anti-clockwise direction.

2. Scaling

   This has the following options:

   - Scale to Fit

     The picture, if there is one, is scaled uniformly so that the logical page containing the picture just fits inside the plotting area.

   - Real Size - View Paper

     This option displays the picture at its real size, within the Output Area, on the paper.

   - Real Size - Position Picture

     This option shows the Output Area relative to the picture on the logical page containing the real size picture. By moving the

area box over the picture, by using the curso and pointer device, a different part of the picture can be selected.

While in this mode, a panel appears with a return option which has to be selected to terminate this mode and return to the options with View Paper selected.

Note: There is no *cancel* option (a *return* option is used to return), and so any operation remains in affect, unless undone by another operation.

12.1.6.1.3 Exit

This causes the utility to terminate.

The initial defaults for a printer class device is a mono print, with normal mapping, on the default paper size for the printer. The print is scaled to fit within the clip limits and the paper orientation is upright.

The initial defaults for a plotter class device is that a full plot is carried out, paper coloured objects are not plotted, and the plotter character set is not used, on the default paper size for the plotter. The plot is scaled to fit within the clip limits and the paper orientation is upright.

12.1.6.2 Line Command

The format of the line command is:

```
PicPrint ((d:)(path)filename(.ext)  (options))
```

where:

1. The file specified is the source file.

2. options can be the following:

    1. /P

        This causes the file to be printed, using any options as defined by the User Interface (see the section, "User Interface").

        If this option is not specified, then the User Interface (see the section, "User Interface") is entered, with the specified file being selected.

If no parameters are specified, the User Interface is entered (see the section, "User Interface").

12.1.6.3 Application Functions

The API provided is as follows:

PicPrint

```
BOOL PicPrint (filename, type)
LPSZ filename;
LONG type;
```

This prints a metafile using the current destination and appearance options.

Parameters:

filename
  An ASCIIZ string giving the name of the metafile.

type   The type of file:

1. Metafile

2. PIF

Returns:

0 Error
1 OK

12.1.7   Naming Conventions and Other Standards

12.1.7.1 Subdirectories

The subdirectories to be used can be defined via the Control Panel (see the section "Control Panel") and are held in the Presentation Manager initialisation file (see the section, "The Initialization File"), which can also be queried by an application. If a subdirectory has not been defined, the following are used:

- \METRES

  Metafile resources

  - xxx.SMB

    Symbol set

  - xxx.LIN

    Line type table

  - xxx.COL

    Color table

- xxx.FNT

  Font
- \PIFRES

  PIF resources:
  - xxx.SMB

    Symbol set
  - xxx.LIN

    Line type table

12.1.7.2 Search Order

Drive searches are as follows:

1. Default drive.
2. Drives in DOS drive letter order.

12.1.7.3 Filename Extensions

Default filename extensions are as follows:

- .MET - metafile
- .PIF - PIF file
- .SMB - symbol set
- .LIN - line type table
- .FNT - font
- .COL - color table
- .PCL - piclist file

Chapter 13

General Functions

13.1   General functions

13.1.1   Program Initialization

Presentation Manager applications are constructed just like any other application for   the target system. The only difference is that the application contains calls upon one or more of the Presentation Manager functions. An important function is the WinInitialize function, as it must be called prior to another Presentation Manager N function. Equally important is the value returned by the WinInitialize function. This value, called the "anchor block" handle, or "hab", must be passed as the first parameter to all other Presentation Manager functions that do not themselves take a handle parameter of any other handle type.

```
typedef UINT HAB;
```

WinInitialize

Format

```
HAB    WinInitialise (options)
HAB    hab;
INT    options;
```

Purpose   This function initializes an application thread for making Presentation Manager system calls. It returns an anchor block handle that is passed as the first parameter to many Presentation Manager system calls, and thus must be the first Presentation Manager system call made by an application thread. Returns NULL if the application thread could not be initialized.

options parameter controls a number of options which affect the whole of the application thread in relation to Presentation Manager functions.

options can take the following values:

NULL   implies default operation of the call. This is the only value supported by Presentation Manager.

WM_ DISABLE
indicates that at window creation all messages are to be handled by the default window process. ie. The application will not receive any messages - see the WinSetMsgInterest call for further details. If this is not specified, the

429

default state when a window is created is
that all messages are enabled.

13.1.1.1 Program Termination.

Just as there is a call to Initialise the Presentation Manager API, there is a
call to Terminate use of the API. However, there is also a message, the
WM_ QUIT message, which is used to indicate to the main loop of the pro-
gram that it is time to terminate.

WM_ QUIT

Format

```
WM_QUIT
lparaml:    OL
lparam2:    OL
```

Description

This message is posted to terminate the applica-
tion. It causes WinGetMsg() to return FALSE,
rather than TRUE as for all other messages.

An application's main loop typically is:

```
while (WinGetMsg(...)) WinDispatchMsg;
...
...
/* application termination code */
```

The loop continues forever until a WM_ QUIT
message is returned by WinGetMsg(). Applications
that call WinPeekMsg() rather than WinGetMSG()
should test explicitly for WM_ QUIT.

Typically, the WM_ QUIT message is posted by
the application when the application quit com-
mand is selected from the Action Bar.

WinTerminate

Format

```
void WinTerminate(hab)
HAB hab
```

Purpose   This function terminates an application thread's
use of the Presentation Manager   the program-
ming interface. This function deallocates all
Presentation Manager resources allocated to the
thread concerned and any subsequent calls to

Presentation Manager functions by the thread will fail.

hab is the anchor block handle returned by the WinInitialize function.

It is good practice to issue this call prior to termination of the program. However, if it is not issued, all Presentation Manager resources allocated to the thread get deallocated when the program terminates - whether normally or abnormally - by Presentation Manager code executed as part of the Exit List processing.

13.1.2 Heap Manager

This section describes the functions for creating, destroying and reorganizing heaps; for allocating, reallocating and freeing memory objects within a heap; and for accessing the base address of a heap.

There are the following heap management functions:

— WinCreateHeap
— WinDestroyHeap
— WinAvailMem
— WinAllocMem
— WinReallocMem
— WinFreeMem
— WinLockHeap
— WinUnlockHeap

A heap is a segment that contains other memory objects allocated and freed using the functions defined in this module.

A heap handle is a 32 bit quantity that uniquely identifies a heap, the segment that contains the heap and the offset within the segment of the beginning of the heap. The interpretation of the heap handle is implementation dependent.

A heap has the following properties:

- it is a segment allocated by DOS, either by default because it is the application's or dynlink package's automatic data segment or it was explicitly allocated with DosAllocSeg.

- it may not be larger than 64k bytes.

- all pointers to memory objects within a heap are 16 bit offsets from the start of that segment. All memory objects are aligned on a ULONG boundary. This means that the contents of the low order 2 bits of a returned pointer are up for grabs for use by the caller. WinAllocMem sets the bits to zero. WinReallocMem and Win-FreeMem ignore the bits, but preserve them in the returned value.

- the heap manager does not remember the size of a memory object allocated within a heap. It is up to the caller to remember the size and specify it whenever reallocating or freeing a heap memory object. The heap manager always rounds a size parameter up to the next multiple of 4 (ULONG alignment); thus the caller does not need to be concerned with the size of any extra space allocated to meet the alignment constraints.

- the contents of memory allocated with the heap manager is undefined (i.e. it does not explicitly set it to all zeros). A debugging version of the heap manager may explicitly set it to all ones in order to help find places where uninitialized memory is used as a pointer or segment address (i.e. DS: FFFF and FFFF: xxxx will typically generate a GP fault).

- when allocating or reallocating a memory object within a heap, DosReallocSeg may be called to expand the segment in order to satisfy the request. The amount of growth is bounded below by the size of the request and bounded above by the minimum growth amount, which is a parameter to the WinCreateHeap function.

- the heap manager maintains an array of free lists indexed by size. This allows the allocator to find a free block quickly. Since the array occupies memory, there is a method for setting the size of the array when the heap is created. The minimum and default size of the array is one entry, called the non-dedicated free list. This entry is the head of a linked list of free blocks, in some undefined order. Additional entries in the free list array are called dedicated free lists. The WinCreateHeap function allows the caller to specify the minimum and maximum object sizes that are to have their own dedicated free lists. Allocating and freeing objects within these minimum and maximum sizes that have their own free list is very quick, as no search is required if there is already a free block of the requested size. A linear search of the non-dedicated free list is performed only if a block of the requested size can't be found on a dedicated free list,

- there is a function provided that returns the size of the largest free block within a heap. It has the option of reorganizing the heap in order to attempt to make a free block that is larger than a specified amount.

- there is no overhead for allocated or free memory objects except for that imposed by the ULONG alignment (maximum of 4 bytes, average of 2 bytes). The only other overhead imposed by the heap manager is the size of the heap control block, which contains the free list array and a few other control words.

- if a zero size object is allocated, it will occupy a minimum of 4 bytes in order to have a unique address to return to the caller.

- the shareability of a heap depends upon the shareability of the segment containing the heap. Heaps contained within an application's data segment are private to that application. Heaps contained within a dynlink package's data segment are either private to each process (instance data segment) or shared across all processes (global data segment). Segments explicitly allocated with DOSALLOCSEG are share or private depending upon the setting of the shareInd parameter to DOSALLOCSEG. Segments allocated by WinCreateHeap are allocated shareable. It should be noted that shared segments are not allowed to shrink, which means that heaps within a shared segment will also not be able to shrink.

- the heap manager makes no attempt to provide mutual exclusion between multiple threads of execution attempting to call the heap manager with the same heap handle. It is the caller's responsibility to insure that this does not occur.

There is a special type of heap, called a moveable heap, that allows the memory objects within the heap to move in order to reclaim fragmented heap space. All heaps are moveable in the sense that the segment that contains a heap can move due to the selector -> physical address mapping provided by the 286 protected mode. Moveable heaps have the following properties:

- the moveable heap attribute is specified at the time the heap is created and lasts until the heap is destroyed.

- allocated objects in a moveable heap have two extra words reserved at the beginning of each object: a handle word and a size word. The offset value returned by WinAllocMem and WinReallocMem is the offset of the first reserved word, the handle word.

- the size word is initialized to the exact size specified on the WinAllocMem call and does not include the 4 bytes allocated for the reserved words or any bytes allocated to meet the ULONG alignment constraints.

- the handle word is initialized to zero when an object is allocated. If it is set to a non-zero value by the caller, then the value must be a 16 bit offset within the segment containing the heap. This offset points to a word, called the handle value word. The low order bit of the handle word must always be zero, as it is used to identify free blocks. This allows the compaction algorithm in WinAvailMem to do a linear scan of the objects in the heap.

- the WinReallocMem and WinAvailMem functions will feel free to move blocks that have a non-zero handle word. Allocated objects whose handle word is zero will be considered fixed and will not move.

- whenever an object with a non-zero handle word moves, the handle value word is updated by the delta amount of the move. By adding a delta instead of storing the new address of the object, it minimizes the constraints upon the type of address in the handle value word. For example, the caller may store the address of the first byte after the two reserved words in the handle value word, in order to hide the reserved words from code that accesses the memory object.

- it is the callers responsibility to reserve space for the handle value words. If that space is within a heap memory object, it is hoped that it is a fixed memory object (i.e. its handle word is zero). Otherwise incorrect results will occur.

- the size parameter to the WinReallocMem and WinFreeMem functions is ignored for objects in a moveable heap, as the value of the size word is used instead.

- objects within a moveable heap can move whenever the WinAvailMem function is called. Since this function is also called by WinAllocMem and WinReallocMem objects can also move when these functions are called as well.

It is important to note why there is a heap manager in the Presentation Manager API when MS OS/2 already provides the Memory Sub-Allocation Package (MSP). MSP has three major drawbacks that make it unsuitable for Presentation Manager applications:

1. It requires that the heap always be at the beginning of a segment, which means that it can't be used to allocate memory out of the space reserved at the end of an application's automatic data segment by the HEAPSIZE keyword in the application's .DEF file. The same is true for the automatic data segment associated with a dynlink package.

2. The implementation of MSP is inefficient as it does not maintain dedicated free lists, and thus is required to do a linear search of its non-dedicated free list whenever a memory object is allocated. Since the free list is kept in address sorted order, the cost of the search can vary. In fact the cost of a free can also vary. Granted this in an implementation restriction, but adding dedicated free lists requires two API changes to support the concept well: ability to control the number of dedicated free lists and an API call to coalesce free blocks, since it can't be done when a memory object is freed as MSP does.

3. It does not provide any mechanism for supporting moveable objects within a heap.

WinCreateHeap

Format

```
HHEAP WinCreateHeap(segHeapBase, cbHeap, cbGrow,
        cbMinDed, cbMaxDed, fOptions)
UINT segHeapBase
UINT cbHeap
UINT cbGrow
UINT cbMinDed
UINT cbMaxDed
UINT fOptions
```

Purpose   This function creates a heap that can used for local memory management.

Parameters

Parameter
   Significance

segHeapBase
   An unsigned integer value that specifies the segment address of the segment to contain the local heap.

cbHeap   An unsigned integer value that specifies the initial size of the heap, in bytes.

cbGrow   An unsigned integer value that specifies the minimum number of bytes to grow the heap by if the heap is too small to satisfy a memory allocation request.

cbMinDed
   An unsigned integer value that specifies the minimum number of dedicated free lists.

cbMaxDed
   An unsigned integer value that specifies the minimum number of dedicated free lists.

rgfOptions
   An unsigned integer value that specifies optional characteristics for the heap. Currently the only option supported is the HM_MOVEABLE option.

Return Value

> The return value, a heap handle, is nonzero if the heap is initialized. Otherwise, it is zero. This heap handle must be passed as the first parameter to all the remaining heap functions.

This function creates a heap that can used for heap allocation using the remaining functions in this module. The first two parameters specify the segment to contain the heap and the size of the heap, in bytes. There are three possible types of segments that can contain a heap:

1. an application's automatic data segment.

2. a dynlink package's automatic data segment.

3. a segment allocated with DosAllocSeg (Public or shared)

In order to accomodate these various targets for heaps, all four possible combinations of the two parameters are used to discriminate between the various options. The combinations and their meaning are:

segHeap/cbHeap	Meaning
0/0	Caller is an application that wants the heap located at the end of its automatic data segment. The size of the heap was specified with the HEAPSIZE keyword in the application's .DEF file to the linker. This function extracts the heap size parameter from the local infoseg and uses that many bytes at the end of the caller's automatic data segment. No reallocation of the data segment occurs, as the DOS loader already reserved the space at the end of the data segment, after the static data was loaded from the .EXE file.
selector/!=0	Caller is a dynlink package that wants a heap placed at the end of its automatic data segment. The cbHeap parameter must be less than or equal to the HEAPSIZE value from the .DEF file that was passed to the dynlink package's initialization entry point in the CX register. Otherwise this function may produce incorrect results.

selector/0	Caller is either an application or dynlink package that has explicitly allocated a segment with DosAllocSeg and wants to put a heap in that segment. The heap is placed at the beginning of the segment and the size of the segment (determined using DosSizeSeg) is the size of the heap.
0/!=0	Caller is either an application or dynlink package that wants a heap of a specific size in a separate segment, but does not want to be bothered with calling DosAllocSeg. See the WinLockHeap function for accessing the base of the segment implicitly allocated by WinCreateHeap when called with this combination of parameters.

The return value is a handle to a heap that must be passed as the first parameter to all the remaining functions in this module. If the return value is zero, then no heap was created, either because of lack of memory in the last case, an invalid selector in the 2nd and 3rd case and no HEAPSIZE parameter in the first case.

The remaining four parameters are used to control how the remaining functions in this module behave.

The cbGrow parameter determines the minimum number of bytes the WinAllocMem and WinReallocMem functions will grow the segment containing the heap if the heap must be grown in order to satisfy a request for memory. If zero is specified for this parameter, then the default value will be 512 bytes.

The cbMinDed and cbMaxDed parameters determine which memory object sizes for the heap will have their own free list. Objects within this size range will incur minimal overhead on an allocate if there is a free block of the requested size on a dedicated free list. The default value of these parameters is 0, which means that no size will have a dedicated free list and all sizes will be on the non-dedicated free list. Allocation requests that cannot be satisfied with a free block from a dedicated free list will do a linear search of the non-dedicated free list until the first free block large enought to satisfy the request is found. The cost of each dedicated free list is an additional word (2 bytes) in the heap control block. Since only sizes that are a multiple of 4 are allowed, the number of freelists will be ((cbMaxDed+3) & ~3) - (cbMinDed & ~3).

The fOptions parameter contains flags that define various
options. Currently the following options are defined:

HM_ MOVEABLE

this bit specifies that the created heap should sup-
port moveable objects. This causes DosAllocMem
to reserve an additional two words at the begin-
ning of each allocated object.

WinDestroyHeap

Format

```
HANDLE WinDestroyHeap(hHeap)
HHEAP hHeap;
```

Purpose   This function destroys a heap that was previously
created with the WinCreateHeap function. If Win-
CreateHeap called DosAllocSeg to allocate space
for the heap, then WinDestroyHeap will call Dos-
FreeSeg to free the allocated segment. Otherwise,
WinDestroyHeap only frees the passed heap han-
dle.

The return value of this function is zero if it is suc-
cessful. Otherwise the return value is the passed
heap handle. Possible reason for failure is an
invalid heap handle. This function does not care if
there are allocated memory objects within the
heap.

This function makes no attempt to insure that the
hHeap being destroyed is not reused by a later call
to WinCreateHeap.

Parameters

Parameter
        Significance

hHeap   The handle to a heap. This handle must
        have been returned from a previous call
        to WinCreateHeap. If this parameter is
        null then this function is noop.

Return Value
        The return value, a heap handle, is NULL if the
        function succeeded. Otherwise this function
        returns the hHeap parameter. This allows the fol-
        lowing idiom for destroying a heap and invalidat-
        ing the variable that contains the heap handle:

```
        HANDLE hHeap;

        hHeap = WinCreateHeap( ... );

   ...

        hHeap = WinDestroyHeap( hHeap );
```

WinAllocMem

Format

```
char * WinAllocMem(hHeap, wBytes)
HHEAP hHeap;
UINT cb;
```

Purpose   This function allocates a memory object in the heap specified by the first parameter and of the size specified in the second parameter. It returns the 16 bit offset from the start of the segment containing the heap, of the allocated memory object. The low order two bits of the returned pointer are always zero. It returns NULL if it is unable to allocate the memory object, either because an invalid heap handle was specified or there was not enough room in the heap for an object of the specified size and it was unable to grow the segment containing the heap by an amount large enough to satisfy the request.

If the passed heap was created with the HM_ MOVEABLE option, then the value of the cb parameter will be remembered in the second reserved word of the allocated block. The returned address will be the address of the first reserved word.

The allocation algorithm first looks in the dedicated free lists, starting with the one whose size is >= the requested size. It proceeds looking in the dedicated free lists, until either it finds the smallest block >= the requested size or it exhausts the dedicated free lists. If no block is found on the dedicated free lists, then it does a linear search of the non-dedicated free list for the first block that satisfies the request (it may not be the smallest free block that would satisfy the request, as that is implementation dependent on how the non-dedicated free list is ordered). Dedicated free lists are ordered on a LIFO basis.

If the free block found is larger than needed to satisfy the request, the extra space is added to the

appropriate free list.

If no free block is found, then this function next attempts to get the space by calling WinAvail-Mem. If that does not generate a free block big enough, it then attempts to grow the segment by the maximum of the size of the request and the minumum growth parameter specified on the Win-CreateHeap call. If that fails, then this function returns NULL.

Parameters

Parameter
Significance

hHeap   A handle to a heap, This handle must have been returned by a previous call to the WinCreateHeap function.

cb   An unsigned short integer value specifying the total number of bytes to allocate.

Return Value

The return value is a short pointer to the allocated memory block if the function is successful. Otherwise, it is NULL. The short pointer is relative to the beginning of the segment that contains the heap.

WinReallocMem

Format

```
char * WinReallocMem(hHeap, pMem, wOldBytes,
        wNewBytes)
HHEAP hHeap;
char *pMem;
UINT cbOld;
UINT cbNew;
```

Purpose   This function is used to grow or shrink a memory object allocated with WinAllocMem. The caller must specify both the old size of the memory object and the new size. If the new size is larger than the old size then this function calls WinAlloc-Mem to allocate the new, larger object, copies cbOld bytes from the old object to the new, frees the old and returns a pointer to the new object (i.e. it will never grow an object in place).

The return value of this function is a 16 bit offset from the start of the segment containing the heap,

of the reallocated memory object. It returns NULL if it is unable to reallocate the memory object, either because an invalid heap handle was specified; there was not enough room in the heap to grow the object to the specified size; or the pMem parameter pointed to memory outside of the bounds of the passed heap.

The low order two bits of pMem are ignored, except they are preserved in the return value of this function, even if the memory object is moved as a result of growing. Except for the two low bits, the value of the pMem parameter must have been returned by either the WinAllocMem function or a previous call to WinReallocMem.

If the passed heap was created with the HM_MOVEABLE option, then the value of the cbOld parameter is ignored and the value in the size word of the allocated object is used instead. When this function is done, the size word will contain the value of the cbNew parameter. If this function had to move the object in order to satisfy the request, then the handle value word will be updated by adding to it the delta amount of the move, in bytes. The returned address will be the address of the first reserved word.

Parameter

Parameter	Significance
hHeap	A handle to a heap, This handle must have been returned by a previous call to the WinCreateHeap function.
pMem	A handle to the memory block to be reallocated.
wOldBytes	An unsigned short integer value specifying the old size of the memory block.
wNewBytes	An unsigned short integer value specifying the new size of the memory block.

Return Value

The return value is a short pointer to the reallocated memory block if the function is successful. Otherwise, it is NULL. The short pointer is relative to the beginning of the segment that contains the heap.

The return value may be different from the pMem parameter, if the block was grown in size and could not be done in place.

WinFreeMem

Format

```
char * WinFreeMem(hHeap, pMem, wBytes)
HHEAP hHeap;
char *pMem;
UINT cbMem;
```

Purpose   This function is used to free a memory object allocated with WinAllocMem. It returns NULL if successful, otherwise it returns the pMem parameter. It can fail either because of an invalid heap handle or an invalid pMem parameter that points outside of the bounds of the heap.

Except for the two low bits, which are ignored, the value of the pMem parameter must have been returned by either the WinAllocMem or WinReallocMem function.

If the passed heap was created with the HM_MOVEABLE option, then the value of the cbMem parameter is ignored and the value of the size word in the allocated object is used instead. If the handle word is non-zero then the contents of the handle value word are set to zero after the object has been freed.

This functions either inserts the passed memory object at the head of the dedicated free list of the given size, or if there is no dedicated free list for that size, it inserts the object into the non-dedicated free list (sort order is implementation specific and undefined).

This function does NOT attempt to coalesce the block being freed with other free blocks. Use the WinAvailMem function to force free blocks to be coalesced.

Parameters

Parameter
     Significance

hHeap   A handle to a heap, This handle must have been returned by a previous call to the WinCreateHeap function.

pMem   A pointer to the memory block to be freed. It must have been returned by a previous call to WinAllocMem or Win-ReallocMem

cbMem   The size of the memory to freed. It must match the allocated size of the block.

Return Value

The return value is NULL if the function is successful. Otherwise, it is equal to pMem.

WinLockHeap

Format

```
LPCH WinLockHeap(hHeap)
HHEAP hHeap;
```

Purpose   When allocation memory out of an application data segment, the short pointers returned are directly useable as offsets relative to DS. When allocating memory out of other segments, the short pointers returned must be dereferenced relative to the beginning of the segment containing the heap. This function returns the address of the beginning of the segment that contains the specified heap.

Parameters

Parameter
    Significance

hHeap   A handle to a heap, This handle must have been returned by a previous call to the WinCreateHeap function.

Return Value

This function returns a far pointer to the beginning of the segment that contains the passed heap.

Note   In the MS-DOS environment, this function locks the segment containing the heap.

WinUnlockHeap

Format

```
BOOL WinUnlockHeap(hHeap)
HHEAP hHeap;
```

Purpose This function is the converse of the WinLockHeap function and marks when the application is done using the segment value returned by WinLock-Heap. Under DOS, this function does nothing. Under MS-DOS it decrements the lock count associated with the moveable segment that contains the heap.

Parameters

Parameter
    Significance

hHeap A handle to a local heap, This handle must have been returned by a previous call to the WinCreateHeap function.

Return Value
This function returns TRUE if the segment is still locked and that another call to this function can be made. It returned FALSE if the segment is now unlocked.

13.1.3   Atom Manager

This section describes the functions used for creating and destroying atom tables; for adding, finding and deleting atoms within an atom table; and for accessing the string name and usage count associated with an atom.

There are the following functions:

```
WinCreateAtomTable
WinDestroyAtomTable
WinAddAtom
WinFindAtom
WinDeleteAtom
WinQueryAtomName
WinQueryAtomLength
WinQueryAtomUsage
```

The atom manager provides a mechnism for converting a string (atom name) into a 16 bit word (atom) that may be used as a constant to represent the string in various application and system data structures. By converting strings to atoms once, you can save space when the same string must be kept in various data structures. It saves time when comparing for a particular string, as you only need to convert the search string to an atom once, then run over your data structures doing word compares with the atoms stored in the data structures.

The atom manager uses an atom table to hold the strings associated with atoms along with the control structures needed to probe the table to see if a string is already there. Atom tables have the following properties:

- maximum length of an atom name is 255 characters. A zero length string is not a valid atom name.

- when searching for an atom name in an atom table, case is significant and the entire string must match (i.e. no substring matching is performed). The search is performed using a hash table. Collisions are resolved using chaining.

- the maximum amount of data that can be stored in an atom table is 64k bytes. This includes any control data needed by the atom manager to manage the atom table (see below).

- the maximum number of strings atoms allowed is 16k. The values of string atoms can range from 0xC000 to 0xFFFF

- associated with each string atom is a usage count that is incremented each time the atom is "added" to the table and decremented each time the atom is "deleted" from the table. This allows multiple users of the same atom string to coexist without destroy each other's atoms.

- this implementation of the atom manager has 6 bytes of overhead per string atom (not including the string itself); plus 2 bytes of overhead for each bucket in the hash table; plus 16 bytes of overhead for the atom table itself.

There is a special kind of atom called an integer atom that has its own set of properties:

- integer atoms can range from 0x0001 to 0xBFFF. The range of integer atoms is disjoint from the range of string atoms and thus the two types of atoms can be intermixed without fear of collisions.

- the string representation of an integer atom is "#ddddd" where the "ddddd" are decimal digits. Leading zeros are ignored. These strings must always be specified in the system code page (850 on a PC).

- there is no usage count or storage overhead associated with an integer atom.

- integer atoms are useful for predefined system constants exported by a dynlink package, since they behave exactly like atoms except that they have no overhead. A good example of where integer atoms are used is in the predefined Presentation Manager window classes. Application defined window class names are strings that are converted into atoms for the purpose of detecting if the same class name is being defined more than once. Thus the predefined window classes implemented by Presentation Manager also need to be expressable as atoms. Making them integer atoms allows the

atoms to be expressed as compile time constants in the Presentation Manager header file; the application can reference these classes and create windows with them without having to have a string constant in their data segment; there is no runtime overhead in the Presentation Manager system for remembering these atoms.

Both types of atoms can be specified via a single far pointer that can be interpreted in one of four ways.

1. lpString -> "string"   ; string atom name

2. lpString -> "#ddddd"   ; integer atom specified as a string

3. lpString -> "!",atom   ; integer or string atom passed indirectly

4. lpString = FFFF:atom   ; integer or string atom passed directly

Note that these representations put a constraint on the first character. Namely that if it matches with the pound sign (#) or exclamation mark (!) in the system code page (850 on a PC) then the remainder of the string will be treated specially.

WinCreateAtomTable

Format

```
HATOMTBL WinCreateAtomTable(cbInitial, cBuckets)
UINT cbInitial;
 UINT cBuckets
```

Purpose

This function creates an empty atom table of the specified size.

This function should be called before any other atom manager function.

Parameters

Parameter
> Significance

cbInitial
> An unsigned short integer value specifying the initial number of bytes to be initially reserved for the atom table. This size is a lower bound on the amount of memory reserved. The amount of memory actually used by an atom table will depend upon the actual number of atoms stored in the table. If this parameter is zero then the size of the atom table will be the minimum size needed to store the atom hash table.

cBuckets

> An unsigned short integer value specifying the size of the hash table used to access the atoms. If this value is zero, then the default value used will be 37. A prime number gives the best results.

Return Value

> The return value, a HANDLE value, is NULL if the function failed. Otherwise it is a handle to the atom table, which must be passed as a parameter to the remaining atom manager functions.

WinDestroyAtomTable

Format

```
HATOMTBL WinDestroyAtomTable(hAtomTbl)
HATOMTBL hAtomTbl;
```

Purpose   This function destroys an atom table created by WinCreateAtomTable.

> This function makes no attempt to insure that the handle to the atom table is not reused by a later call to WinCreateAtomTable.

Parameters

Parameter
> Significance

hAtomTbl

> A handle to an atom table. This handle must have been returned from a previous call to WinCreateAtomTable. If this parameter is null then this function is no-op.

Return Value

> The return value, an atom table handle, is NULL if the function succeeded. Otherwise this function returns the hAtomTable parameter. This allows the following idiom for destroying an atom table and invalidating the variable that contains the atom table handle:

```
HATOMTBL hAT;

hAT = WinCreateAtomTable( ... );
```

   ...

```
hAT = WinDestroyAtomTable( hAT );
```

WinAddAtom

Format

```
ATOM WinAddAtom(hAtomTbl, lpszAtomName)
HATOMTBL hAtomTbl;
LPCH lpszAtomName;
```

Purpose   This function adds an atom name (pointed to by lpszAtomName) to an atom table (given by hAtomTbl). If the atom name is already in the atom table, then it just increments the usage count associated with the atom. Otherwise adds the atom to the table and initializes its usage count to one. In either case this function returns the atom that represents the passed atom name. It returns zero if an invalid atom table or atom name was specified.

If the lpszAtomName parameter points to a string of the form "#ddddd" then this function converts the ASCII decimal digits into a 16 bit integer and if it is a valid integer atom returns that atom, without actually modifying the atom table.

If the lpszAtomName parameter points to an exclamation mark ("!") then the word following that character is assumed to be an atom. If it is an integer atom, it is returned. If it is not an integer atom AND it is a valid atom for the given atom table (i.e. it has an atom name and usage count associated with it) then the usage count is incremented and the atom is returned. Otherwise zero is returned.

If the high order word of the lpszAtomName parameter is 0xFFFF then the low order word is treated as an atom and processed in the same manner as the atom described in the previous paragraph.

Parameters

Parameter
          Significance

hAtomTbl
          A handle to an atom table. This handle must have been returned from a previous call to WinCreateAtomTable.

448

lpszAtomName

>A long pointer to the character string to be added to the table. The string must be an null-terminated ASCII string. If the string begins with a "#" character, then the ASCII digits that follow are converted into an integer atom. If the string begins with a "!" character, then the next two bytes are intepreted as an atom.

>If the high order word of this parameter is -1 then the low order word is an atom. If it is an integer atom, then that atom is returned. Otherwise the reference count associated with that atom is incremented.

Return Value

>The return value, a unsigned short integer value, is the atom that is associated with the passed string. Otherwise, it is NULL.

Notes

The atom values returned by WinAddAtom are in the range C000 to FFFF (hexadecimal). The range 0001 to BFFF (hexadecimal) is reserved for integer atoms.

WinFindAtom

Format

```
ATOM WinFindAtom(hAtomTbl, lpszAtomName)
HATOMTBL hAtomTbl;
LPCH lpszAtomName;
```

Purpose

This function is identical to the WinAddAtom function with two exceptions:

- if the atom name is not found in the table, it is NOT added to the table and zero is returned as the value of this function.

- if the atom name is found in the table, the usage count is NOT incremented.

Since integer atoms do not have a usage count and do not actually occupy memory in the atom table, this function IS identical to WinAddAtom with respect to integer atoms.

Parameters

Parameter
  Significance

hAtomTbl
  A handle to an atom table. This handle
  must have been returned from a previous
  call to WinCreateAtomTable.

lpszAtomName
  A long pointer to the character string to
  be added to the table. The string must
  be an null-terminated ASCII string. If
  the string begins with a "#" character,
  then the ASCII digits that follow are
  converted into an integer atom. If the
  string begins with a "!" character, then
  the next two bytes are intepreted as an
  atom and that atom is returned.

  If the high order word of this parameter
  is -1 then the low order word is an atom
  and that atom is returned.

Result   The return value, a unsigned short integer value, is
          the atom associated with the given string. It is
          NULL if the string is not in the table.

WinDeleteAtom

Format

```
UINT WinDeleteAtom(hAtomTbl, atom)
HATOMTBL hAtomTbl;
ATOM atom;
```

Purpose   If the passed atom is an integer atom, zero is
          returned. If it is not an integer atom AND it is a
          valid atom for the given atom table (i.e. it has an
          atom name and usage count associated with it)
          then the usage count is decremented and zero is
          returned. If the usage count has been decremented
          to zero then the atom name and usage count are
          removed from the atom table. If either the atom
          table or atom are invalid, the atom is returned.

          Returning zero to indicate success allows the use of
          the idiom described with WinDestroyAtomTable.

Parameters

Parameter
>   Significance

hAtomTbl
>   A handle to an atom table. This handle
>   must have been returned from a previous
>   call to WinCreateAtomTable.

atom
>   An unsigned short integer value specify-
>   ing the atom to be deleted.

Return Value
>   The return value, a unsigned short integer value, is
>   NULL if the function is successful. It is equal to
>   atom if the function failed and the atom has not
>   been deleted.

WinQueryAtomName

Format

```
int WinQueryAtomName(hAtomTbl, atom, lpszBuffer,
        cchBufferMax)
HATOMTBL hAtomTbl;
ATOM atom;
LPCH lpszBuffer;
UINT cchBufferMax;
```

Purpose   This function returns the atom name associated
with the passed atom. For integer atoms, the for-
mat of the string is "#ddddd" where the "ddddd"
are decimal digits in the system code page (850 on
a PC). No leading zeros are generated so the length
can range from 3 to 7 characters. The lpszBuffer
parameter points to where to return the atom
name. The cchBufferMax parameter specifies the
maximum number of characters that can be stored
in the buffer (including the terminating null byte).

The return value of this function is the actual
number of characters stored in the buffer (not
including the terminating zero byte). If the atom
table or atom is invalid then this function returns
zero.

Parameters

Parameter
>   Significance

hAtomTbl
>   A handle to an atom table. This handle
>   must have been returned from a previous

call to WinCreateAtomTable.

atom   An unsigned short integer value identifying the character string to be retrieved.

lpszBuffer
A far pointer to the buffer to receive the character string.

cchBufferMax
An unsigned short integer value specifying the maximum size, in bytes, of the buffer.

Return Value
The return value is an unsigned short integer value specifying the actual number of bytes copied to the buffer. It is 0 if the specified atom is not valid.

WinQueryAtomLength

Format

```
int WinQueryAtomLength(hAtomTbl, atom)
HATOMTBL hAtomTbl;
ATOM atom;
```

Purpose   This function returns the length of the atom name associated with the passed atom. For integer atoms, the length is always 6. If the atom table or atom is invalid then this function returns zero.

The purpose of this function is to allow an application to determine how big a buffer to passed to the WinQueryAtomName function. The returned length does not include the terminating null byte for a zero terminated string.

Parameters

Parameter
Significance

hAtomTbl
A handle to an atom table. This handle must have been returned from a previous call to WinCreateAtomTable.

atom   An unsigned short integer value identifying the atom whose length is to be returned.

Return Value
> The return value is a short integer value that is the length of the string associated with atom. It is 0 if the specified atom is not valid. Integer atoms always return a length of 5.

WinQueryAtomUsage

Format

```
UINT WinQueryAtomUsage(hAtomTbl, atom)
HATOMTBL hAtomTbl;
ATOM atom;
```

Purpose   This functions returns the usage count associated with the passed atom. For integer atoms it returns 0xFFFF. If the atom table or atom is invalid then this function returns zero.

Parameters

Parameter
> Significance

hAtomTbl
> A handle to an atom table. This handle must have been returned from a previous call to WinCreateAtomTable.

atom
> An unsigned short integer value identifying the atom whose usage count is to be returned.

Return Value
> The return value is an unsigned short integer value that is reference count of the atom. It is 0 if the specified atom is not valid. It is 0xFFFF for integer atoms.

13.1.4   File Functions

This section describes functions used for creating and opening files. There are the following functions:

```
WinOpenFile
WinGetTempDrive
WinGetTempFilename
```

WinOpenFile

Format

```
int WinOpenFile(lpFileName, lpReOpenBuff, wStyle)
```

Purpose   This function creates, opens, reopens, or deletes a file.

Parameters

Parameter
        Signifigance

lpFileName
        A long pointer to a null-terminated character string specifying the name of the file to be opened. The string must consist of characters from the ANSI character set.

lpReOpenBuff
        A long pointer to a data structure having OFSTRUCT type. The structure receives information about the file when the file is first opened, and is used in subsequent WinOpenFile calls to refer to the open file.

wStyle   An unsigned short integer value specifying the action to take. It can be combinations of the following:

OpenFile Styles
        Value

OF_READ
        Opens the file for reading only.

OF_WRITE
        Opens the file for writing only.

OF_READWRITE
        Opens the file for reading and writing.

OF_CREATE
        Creates the file if it does not already exist.

OF_REOPEN
        Opens the file using information in the reopen buffer.

OF_EXIST
> Creates or opens the file, then closes it. Used to test for file existence.

OF_PARSE
> Fills the OFSTRUCT structure but carries out no other action.

OF_PROMPT
> Displays a dialog box that prompts the user for permission to create a file if the requested file does not exist.

OF_CANCEL
> Adds a Cancel button to the OF_PROMPT dialog box. Pressing the Cancel button directs WinOpenFile to return a file-not-found error.

OF_VERIFY
> Verifies that the date and time of the file are the same as when it was previously opened. Useful as an extra check for read-only files.

OF_DELETE
> Deletes the file.

These styles can be combined using the logical OR operator.

Return Value
> The return value is a DOS file handle if the function is successful. Otherwise, it is -1.

Notes
> To close the file after use, use the DosClose system call.
>
> If the file name does not contain a drive or path specification and it is not found in the current directory, then this function will search all of the directories in the PATH environment variable and open the first one that is found, returning the full path name in the openbuff structure.
>
> The OF_CREATE flag always directs WinOpenFile to create a new file. If the file already exists, it is truncated to zero length.

WinGetTempDrive

Format

```
UCHAR WinGetTempDrive(cDriveLetter)
```

Purpose This function takes a drive letter or zero and returns a letter specifying the optimal drive for a temporary file. The optimal drive is the disk drive that can provide the best access time during disk operations with a temporary file.

The function returns the drive letter of a hard disk if the system has one. Otherwise, if cDriveLetter is zero, it returns the drive letter of the current disk, or if cDriveLetter is a letter, it returns the letter of that drive or one that exists.

Parameters

Parameter
    Significance

cDriveLetter
    A byte integer value specifying a disk drive letter, for example, A for disk drive A.

Return Value
The return value is a byte integer value specifying the optimal disk drive for temporary files.

WinGetTempFileName

Format

```
int WinGetTempFileName(cDriveLetter, lpPrefixStrin
lpTempFileName)
```

Purpose This function creates a temporary filename with the following form:

```
<drive>:\<path>\~<prefix><uuuu>.TMP
```

where drive is the drive letter specified by cDriveLetter, path is the pathname of the temporary file (either the root directory of the specified drive or the directory specified in the TEMP environment variable), prefix is all letters (up to the first three) of the string pointed to by lpPrefixString, and uuuu is a hexadecimal representation of the number specified by wUnique.

Parameters

Parameter
Significance

cDriveLetter
A byte integer specifying the suggested drive for the temporary filename. If cDriveLetter is 0, the default drive is used.

lpPrefixString
A long pointer to a null-terminated string to be used as the temporary filename prefix. If this string begins with a drive letter and/or a path specification, then they will override the defaults that would have been chosen by this function.

wUnique
An unsigned short integer.

lpTempFileName
A long pointer to a buffer where the temporary filename is stored. It must be large enough to hold the largest legal path name.

Return Value
The return value, a short integer value, is the unique numeric value used in the temporary filename. If a nonzero value was given for the wUnique parameter, nUniqueNumber is the same number.

Notes   WinGetTempFileName uses the suggested drive letter for creating the temporary filename except in the following cases:

1.

If a hard disk is present, WinGetTemp-FileName always uses the drive letter of the first hard disk.

2.

If a TEMP environment variable is defined and its value begins with a drive letter, that drive letter is used.

If the TF_ FORCEDRIVE bit of cDriveLetter is set, the above exceptions do not apply. The temporary filename will always be created in the current directory of the drive specified by

cDriveLetter, regardless of the presence of a hard disk or the TEMP environment variable.

If wUnique is zero, WinGetTempFileName attempts to form a unique number based on the current system time. If a file with the resulting filename already exists, the number is increased by one and the test for existence is repeated. This process continues until a unique filename is found; WinGetTempFileName then creates a file by that name and closes it.

No attempt is made to create and open the file when wUnique is nonzero.

13.1.5   Presentation Manager API Error Reporting

If any application thread is to make a Presentation Manager call, it must call the WinInitialise function. This initialises a Presentation Manager instance and returns an anchor block. Any Presentation Manager errors that the application encounters after this initialisation are stored in the anchor block. The application may query the error codes with the WinGet-LastError or WinSetLastError calls.

Note that a successful Presentation Manager call does NOT clear the error code. If the application requires to know whether a specific call was successful, it should set the error code to zero with the SetLastError call before making the api call in question.

Note also that the application may both query and then zero the error code with the SetLastError function.

In multiple thread programmes where there are multiple anchor blocks, errors are always stored in the anchor block created by the WinInitialise call of the thread making the Presentation Manager call. Thus, whe calling Query/SetLastError, the application is responsible for specifying the correct anchor block handle for the thread making the call.

If an error occurs on api call, then the value of any returned parameter except the function value (which may be inspected to determine whether an error occurred), is undefined.

WinGetLastError

Format

```
UINT WinGetLastError (hab)
HAB hab;
```

Purpose   This function returns the error code that was set either by a last Presentation Manager call which failed in some way or by a call to the WinSet-LastError function, whichever occurred last. It also zeros the current error code.

Parameters

Parameter
     Significance

hab   The anchor block handle of the thread making the call - this is the value returned by the WinInitialize function.

Return Value
     The return value is an unsigned short integer value that is the last error code.

WinGetErrorInfo

Format

```
LPERRINFO  WinGetErrorInfo( hab )
HAB        hab;
```

Description
     This function returns a long pointer to a block containing information about the previous error code for the current thread. It returns NULL if there is no error information available. The format of the block is defined by the ERRINFO structure.

```
typedef struct tagERRINFO {
    WORD cbFixedErrInfo;
    ERRORID idError;
    WORD cLevels;
    WORD prgLevel;
    WORD pBinaryData;
} ERRINFO;
```

```
typedef ERRINFO far *LPERRINFO;
```

The fields have the following contents:

cbFixedErrInfo
     number of bytes of fixed data at the beginning of the error information block (currently 12). If this field is different from sizeof(ERRINFO) then obviously your code is running on a version of the system that supports less or more than

the system you compiled for.

idError   error ID associated with the error (will match the last non- zero value returned by WinGetLastError).

pBinaryData
is a 16 bit offset to implementation dependendant data. On the PC implementation, this field points to a 16 bit OS/2 error code that is related to this Presentation Manager error.

cLevels   number of levels of detail. It is the number of entries in the array of words pointed to by the following field. This field is never zero (i.e. there is always at least one level of detail).

prgLevel
is a 16 bit offset to an array of 16 bit offsets to null terminate strings. Each string is a printable message that offers varying levels of information. The first level is the least amount of detail and the remaining levels offer more and more detail.

The first level of detail is always an error message string, in the following format:

```
xxxnnnnns message text

where
xxx    is predefined product identifier
nnnnn  is the message number
s      is the message severity letter
       W = Warning
       E = Error
       S = Severe
       U = Unrecoverable
```

and the contents of message text is determined by the format options, format string and insert parameters passed to WinSetErrorInfo function. If there is no format string for the error code, then it will default to "*** unknown error code ***".

The remaining detail levels are optional and implementation dependent. The PC implementation will store the register contents at the time of the error in the second level of detail. The third level of

detail will contains a call backtrace from the point of the error.

The strings are all terminated by a zero byte. All lines are terminated with the carriage return, linefeed character sequence.

Effects: This function allocates a single private segment to contain the ERRINFO block. All of the long pointer to string fields are pointers to memory within that segment.

Warnings:
The memory allocated by this function is not freed until the returned pointer is passed to the Win-FreeErrorInfo API call.

Like WinGetLastError, WinGetErrorInfo will release any saved error information after formatting the error message. If two calls are made to WinGetErrorInfo, without any intervening calls, the second call will return NULL, since the saved error information was consumed by the first call.

WinFreeErrorInfo

Format

```
VOID WinFreeErrorInfo( hab, lpErrorInfo )
HAB      hab;
LPERRINFO lpErrorInfo;
```

Description.
This function frees the memory occupied by the passed error information block. he previous error code for the current thread. It returns NULL

The 32 bit values taken and returned by WinGetLastError and WinSetErrorInfo are defined as follows:

```
typedef ULONG ERRORID;

#define MAKEERRORID( sev, errc ) (ERRORID) (MAKEULONG( (sev), (errc) )
```

The reason the severity is separate from the error code is because it allows the same error code to have two different severity levels, based on context. For example, an invalid parent Window handle passed to WinCreateWindow is a severe error, but an invalid Window handle passed to WinDestroyWindow is just an error. Another advantage of keeping them separate is that most applications will not look at the severity code. This allows them to write the following code sequence, taking advantage of the

efficiences of 16 bit compares:

```
UINT errCode;

if (!(p = WinAllocMem( hHeap, 24 ))) {
errCode = (UINT)WinGetLastError( hab );
if (errCode == WINERR_HEAP_OUT_OF_MEMORY) {
}
else
if (errCode == WINERR_HEAP_MAX_SIZE_REACHED) {
}
else {
}
}
```

13.1.6   General DOS Related Support Functions

This section describes various DOS related support functions.

There are the following functions:

- WinCatch
- WinThrow
- WinQueryVersion
- WinReportError

13.1.6.1   Catch and throw

WinCatch and WinThrow are functions that allow an application to remember a point in the call stack hierarchy and then return to that point from a point lower down in the hierarchy. This is typically useful for dealing with error conditions. For example:

```
CATCHBUF AllocCatchBuf;

proc1() {
    int errorCode;

    if (errorCode = WinCatch( &AllocCatchBuf )) {
        /* cleanup, possibly display error message, etc. */
        return;
        }

    ...

    proc2(...);     /* Call another procedure */

}
```

```
proc2() {
    PMEM p;

    if (!(p = WinAllocMem( 0, 12 ))) {
        WinThrow( &AllocCatchBuf, (WORD)WinGetLastError( hab ));
        /* WinThrow does not return */
        }

}
```

In the above example, if WinCatch and WinThrow were not used, the error condition detected in proc2 would have to be passed back to proc1 via the return code value of proc2. This function becomes more valuable as the number of levels of procedure call between the catch and throw increases.

Note also that it is the responsibility of the routine that receives the catch to free up any resources that had been allocated between the issuance of the catch and the throw.

WinCatch

Format

```
int WinCatch(lpCatchBuf)
```

Purpose   This function catches the current execution environment and copies it to the buffer referenced by lpCatchBuf. The buffer can then be used by the WinThrow function to restore the execution environment later. The execution environment includes the state of all system registers and the instruction counter.

Parameters

Parameter
     Significance

lpCatchBuf
     A long pointer to a buffer having CATCHBUF type.

Return Value
     The return value is a short integer value, is zero if the environment is copied to the buffer.

WinThrow

Format

```
void WinThrow(lpCatchBuf, nThrowBack)
```

Purpose   This function restores the execution environment
to the values saved in the buffer referenced by
lpCatchBuf. Execution then continues in the Win-
Catch function that copied the environment to
lpCatchBuf. The nThrowBack parameter is passed
as the return value to the WinCatch function. It
can be any nonzero integer value.

Parameters

Parameter
        Significance

lpCatchBuf
        A long pointer to a buffer having
        CATCHBUF type.

nThrowBack
        A short integer to be returned by the
        WinCatch function.

Return Value
        None.

WinQueryVersion

Format

```
WORD WinQueryVersion(hab)
HAB hab;
```

Purpose   This function returns the current version identifier
of Presentation Manager.

Return Value
        The return value is an unsigned short integer value
        specifying the major and minor version number of
        Presentation Manager. The high-order byte is the
        minor version (revision) number; the low-order
        byte is the major version number.

WinReportError

Format

```
WinReportError(hCurrent, errCode, bFatal,
        lpDebugString)
```

Purpose This function is used to report a potentially fatal problem to the user. It will display the error code and string and read a response from the user.

Parameters

Parameter

Significance

hCurrent

The value return by the WinInitialize function.

errCode An unsigned short integer that is the error code to display.

bFatal A boolean value that is true if the error is a fatal one and false otherwise.

lpDebugString

A long pointer to a character string. The string must be a null-terminated ASCII string.

Return Value

This function has no return value. It does NOT return if the bFatal parameter is TRUE.

Chapter 14

Multi-Process and
Multi-Thread Applications

14.1 Rules and Guidelines for Complex Applications.

Complex applications are defined as those applications which use the Presentation Manager API along with other advanced features of DOS such as multi processing. The following sections define rules that such applications *must* follow and also some guidelines that should make it easier to program such applications.

14.1.1 Queueless Processing Threads.

It is possible for a process thread which has not allocated a Presentation Manager Input Queue to use many of the functions of the Presentation Manager API. However, it cannot use *all* of the functions. In general terms, functions which *SEND* messages or access the Input Queue can not be used by a thread without a queue.

The functions that *CAN* be used by a thread with no queue include:

- Heap Manager functions
- Atom Manager functions
- System Information functions
- Resource functions
- DOS Related Support functions.
- Graphics (Gpi) functions

Some of the Windowing (Win...) and Shell (She...) functions can *NOT* be used by a thread with no Input Queue, although many of them can. The following is a list of the calls that can not be used:

- WinBeginPaint
- WinCalcFrameRect
- WinCheckWindowLockCount
- WinCloseClipbrd
- WinCreateCaret
- WinCreateCursor
- WinCreateDlg
- WinCreateFrameControls

— WinCreateMenu
— WinCreateStdWindow
— WinCreateStdWindowIndirect
— WinCreateWindow
— WinCreateWindowDC
— WinDefWindowProc
— WinDestroyCaret
— WinDestroyCursor
— WinDestroyMsgQueue
— WinDestroyWindow
— WinDismissDlg
— WinDispatchMsg
— WinDlgBox
— WinEmptyClipbrd
— WinEnableWindow
— WinEndPaint
— WinEnumClipbrdFmts
— WinEnumDlgItem
— WinEnumWindow
— WinExcludeUpdateRgn
— WinExitWindows
— WinFlashWindow
— WinFormatFrame
— WinGetAccelTable
— WinGetCaretInfo
— WinGetClassInfo
— WinGetClipbrdData
— WinGetClipbrdFmtInfo
— WinGetClipbrdOwner
— WinGetCursorInfo
— WinGetCursorPos
— WinGetDlgItemInt

— WinGetKeyState

— WinGetMsg

— WinGetMsgPos

— WinGetMsgTime

— WinGetPhysKeyState

— WinGetQueueStatus

— WinGetSeparator

— WinGetUpdateRect

— WinGetUpdateRgn

— WinGetWindow

— WinGetWindowLockCount

— WinGetWindowProcess

— WinGetWindowText

— WinIndicatePossibleDeath

— WinInitialize

— WinInSendMsg

— WinInvalidateRgn

— WinIsThreadActive

— WinLoadCursor

— WinLoadDlg

— WinLoadMenu

— WinLockScreen

— WinLockVisRgns

— WinLockWindow

— WinMapDlgPoints

— WinMessageBox

— WinOpenClipbrd

— WinPeekMsg

— WinProcessDlg

— WinProcessDlgMsg

— WinProcessSysCommand

— WinRegisterWindowDestroy

— WinReportError

— WinRestrictCursor

— WinScanToChar

— WinScanToVirtual

— WinScrollWindow

— WinSendDlgItemMsg

— WinSendMsg

— WinSetAccelTable

— WinSetActiveWindow

— WinSetCapture

— WinSetClipbrdData

— WinSetClipbrdOwner

— WinSetClipbrdViewer

— WinSetCursor

— WinSetCursorPos

— WinSetDlgItemInt

— WinSetFocus

— WinSetMsgInterest

— WinSetMultipleWindowPos

— WinSetParent

— WinSetSysModalWindow

— WinSetWindowPos

— WinSetWindowText

— WinShowCaret

— WinShowCursor

— WinShowWindow

— WinStartTimer

— WinStopTimer

— WinSubclassWindow

— WinSubstituteStrings

— WinTimeoutSendMsg

— WinTranslateAccel

— WinUnlockWindow

— WinUpdateWindow

— WinValidateRect

— WinValidateRgn

— WinWaitForHotKey

— WinWaitForHungProcess

— WinWaitForSGChange

— WinWaitMsg

— WinWindowFromPoint

14.1.2 Multi-process and multi-thread applications

This section specifies the constraints on an application that makes use of multiple DOS processes or threads.

The following table is a list of the objects which may be created by a Presentation Manager application, and it defines whether each object is accessible to threads or processes that are different to the creating thread/process. i.e. the table defines whether a handle returned to the application by Presentation Manager in one process may be used by the application in another process.

Presentation Manager Object	Another Thread		Another Process	
Window	y	(1, 3)	y	
Dialog boxes				
Menu windows				
GPI Presentation Space (non cached)	y		n	
GPI Presentation Space (cached)	y	(2)	y	(2)
AVIO Presentation Space	y		n	
Input queue	n	(3)	n	
Device Context	y		y	
Private Window class	y		n	
Public Window class	y		y	
Timer	y	(4)	y	
Cursor (or Pointer)	y		y	
Bitmap	y		y	
Font/symbol set	y		y	
Local heap	y		y/n	(5)
Atom Table	y		y	
Region	y		y	
Metafile	y		y	

Notes

473

1. Only the thread that created the queue and the window will receive
    messages for the window. The windowproc or dialogproc will alway
    run in the context of the thread that created the queue and the
    window.

2. The thread/process that obtained the ps from the cache may access
    it.

3. Winprocessdlgmsg() assumes that all of the dialog items were
    created by the thread that is issuing WinProcessdlgmsg().

4. Timer messages are posted only to the queue associated with the
    thread that created the timer.

5. Depends on the shareability of the segment in which the heap was
    created.

Index