Operating System/2™ Programming Tools and Information Version 1.2

Presentation Manager Programming Reference Volume 1

64F0276

Operating System/2™
Programming Tools
and Information
Version 1.2

Presentation Manager Programming Reference Volume 1

First Edition (September 1989)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

Special Notices

The following names, used in this publication, are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM
Operating System/2
OS/2
Presentation Manager
Proprinter
Systems Application Architecture.

The following names, used in this publication, are trademarks or registered trademarks of other companies:

PostScript	Adobe Systems Incorporated
Windows	Microsoft Corporation.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license enquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Armonk, NY 10504.

About this Book

The Operating System/2 (OS/2) Version 1.2 *Presentation Manager Programming Reference* is a detailed technical reference, in two volumes, for application programmers creating programs using Presentation Manager.

For information about the Control Program, see the OS/2 Version 1.2 *Control Program Programming Reference*.

Chapter 1 contains important information. You should read it before using this book or the associated bindings references.

This reference does not give guidance on how to use the calls, nor does it contain information about how the calls are related to each other. It is intended to be used in conjunction with the OS/2 Version 1.2 *Programming Guide*.

Prerequisite Knowledge

The Programming Tools and Information library is intended for professional application developers knowledgeable in at least one programming language in which OS/2 programs can be written. The information in the Programming Tools and Information library assumes you are new to programming with OS/2 and the Presentation Manager. You should understand the OS/2 services available to users. Further information on these can be obtained from the OS/2 product library.

Related Publications

The OS/2 Version 1.2 *Programming Overview* introduces the programming concepts that you should understand before you begin developing applications to run on OS/2, and describes the set of books, tools, programming aids, and sample programs that make up the OS/2 Programming Tools and Information Library.

Bindings for the IBM C/2, IBM COBOL/2, IBM FORTRAN/2 and IBM Macro Assembler/2, languages are provided in their respective bindings references in the same order that they appear this reference. (For simplicity, C/2, COBOL/2, FORTRAN/2, and Macro Assembler/2 are referred to as "C", "COBOL", "FORTRAN", and "MASM".)

Details of publications for OS/2 Version 1.2 and related products are shown in the library diagram on page vi.

OS/2 Product

IBM Operating System/2
Standard Edition
Version 1.2

Getting Started
Using Advanced Features
Product Information

6024926 3.5″ diskette
6024930 5.25″ diskette

OS/2 Programming Tools and Information

IBM Operating System/2
Version 1.2

Installation
Programming Overview
Programming Guide
Building Programs
Dialog Tag Language
Guide and Reference
Dialog Manager Guide
and Reference
Dialog Manager and
Dialog Tag Language
Reference Summary
Programming Reference
(3 books)
Bindings Reference for
Presentation Manager
(4 books for COBOL/2,
FORTRAN/2, C/2, and
Macro Assembler/2)
I/O Subsystems and
Device Support
(2 books)
Systems Application
Architecture
Common User Access:
Advanced Interface
Design Guide

6024929

Separate Order (no charge)

Keyboards and Code Pages

6280345

Available Separately

Command Reference

6024928

Service Coordinator's
Guide

15F2214

Programming Languages

IBM Basic Compiler/2
Version 1.0 6280179

IBM Macro Assembler/2
Version 1.0 6280181

IBM Pascal Compiler/2
Version 1.0 6280183

IBM FORTRAN/2
Version 1.02 6280185

IBM COBOL/2
Version 1.0 6280207

IBM C/2
Version 1.1 6280284

Systems Application Architecture

An Overview

GC26-4341

Writing Applications:
A Design Guide

SC26-4362

Common User Access:
Advanced Interface
Design Guide
SC26-4582

Common User Access:
Basic Interface Design
Guide
SC26-4583

Common Programming Interface

C Reference - Level 2

SC09-1308

COBOL Reference

SC26-4354

Dialog Reference

SC26-4356

FORTRAN Reference

SC26-4357

Procedures Language
Reference
SC26-4358

Presentation Reference

SC26-4359

Organization of this Book

This book is in two volumes. The contents of each volume are as follows:

Volume One (Data Types and Function Calls)

Chapter 1, "Introduction"

This chapter contains important information about the following:

- Notation conventions
- Conventions used in function descriptions
- Error severities.

You should read it before using this book, or the associated bindings references.

Chapter 2, "Data Types"

Chapter 3, "Device Function Calls"

Chapter 4, "Graphics Function Calls"

See page 4-1 for general information about this set of calls and how they are related to each other.

Chapter 5, "Picture Function Calls"

Chapter 6, "Profile Function Calls"

Chapter 7, "Spooler Function Calls"

Chapter 8, "Advanced Video Function Calls"

Chapter 9, "Window Function Calls"

See page 9-1 for information about how these calls are related to each other.

Volume Two (Related Information)

Chapter 10, "Functions Supplied by Applications"

Procedures and hooks; see page 10-1 for general information about these functions.

Chapter 11, "Introduction to Message Processing"

Chapter 12, "Default Window Procedure Message Processing"

Chapter 13, "Button Control Window Processing"

Chapter 14, "Entry Field Control Window Processing"

Chapter 15, "Frame Control Window Processing"

Chapter 16, "List Box Control Window Processing"

Chapter 17, "Menu Control Window Processing"

Chapter 18, "Multi-Line Entry Field Control Window Processing"

Chapter 19, "Prompted Entry Field Control Window Processing"

Chapter 20, "Scroll Bar Control Window Processing"

Chapter 21, "Static Control Window Processing"

Chapter 22, "Title Bar Control Window Processing"

Chapter 23, "Clipboard Messages"

Chapter 24, "Dynamic Data Exchange Messages"

Chapter 25, "Help Manager Messages"

Chapter 26, "Resource Files"

Chapter 27, "Graphics Orders"
This chapter lists the set of graphics orders used by the Graphics Presentation Interface (GPI) calls.

Chapter 28, "Code Pages"
This chapter details the available code pages, together with an example of each one. Information about double-byte character sets is also included.

Appendix A, "Error Explanations"

Appendix B, "Standard Bit-Map Formats"

Appendix C, "The Font-File Format"

Appendix D, "Format of Interchange Files"

Appendix E, "Initialization File Information"

Appendix F, "Virtual Key Definitions."

Data Types and Function Calls

Chapter 1. Introduction

This chapter contains important information. You should read it before using this book or the associated bindings references.

The purpose of this reference is to give important information about function calls, messages, constants, and so on. The calls are described in a *metalanguage*, so that the information is independent of the language being used.

The main feature of a metalanguage is that data types are not given for any specific language. To program in a particular language, you should use the appropriate bindings reference.

Notation Conventions

The following notation conventions are used in this reference.

NULL The term NULL applied to a parameter is used to indicate the presence of the parameter, but with no value.

Implicit Pointer If no entry for a data type 'Pxxxxxxx' is found in Chapter 2, "Data Types," then it is implicitly a pointer to the data type 'xxxxxxx'.

Constant Names All constants are written in uppercase. Where applicable, constant names have a prefix derived from the name of a function, message, or idea associated with the constant. For example:

WM_CREATE	Window message
SV_CXICON	System value
CF_TEXT	Clipboard format.

In this reference, a set of constants with the same prefix is written as follows:

Window message	WM_*
System value	SV_*

and so on.

Conventions Used in Function Descriptions

The documentation of each function contains these sections:

Function name The function name, listed in alphabetic order of 'C' (long) name together with the English name. This is at the top of each page followed by a brief description of the function.

Where applicable, functions are marked 'SAA'. This indicates a portable function described in the *CPI Presentation Reference* (part of the Systems Application Architecture (SAA) subset).

Parameters Each parameter is listed with its data type and a brief description.

There are four kinds of parameter:

Input Specified by the programmer.

Output Returned by Presentation Manager.

Input/Output Specified by the programmer and modified by the Presentation Manager.

Return The return parameter shown in *italics*, together with possible errors, or TRUE/FALSE indicators if a Boolean function.

A list of possible errors (where appropriate) is included in this section. Some functions do not have error messages.

Note: Data types are given in a *metalanguage*. For further details, see above.

Remarks Additional information about the function, where required.

Programming Note: The functions in this book are named in mixed-case for readability, but are known to the system as uppercase character strings. For example, the function "GpiBeginArea" is actually the external name "GPIBEGINAREA."

If you are using a compiler that generates a mixed-case external name, you should code the OS/2 function calls in uppercase.

Error Severities

Each of the error conditions given in the list of errors for each call falls into one of these areas:

Warning	The function detected a problem, but took some remedial action that enabled the function to complete successfully. The return code in this case indicates that the function completed successfully.
Error	The function detected a problem for which it could not take any sensible remedial action. The system has recovered from the problem, and the state of the system with respect to the application remains the same as at the time when the function was requested. The system has not even partially executed the function (other than reporting the error).
Severe Error	The function detected a problem from which the system could not reestablish its state, with respect to the application, at the time when that function was requested; that is, the system partially executed the function. This, therefore, necessitates the application performing some corrective activity to restore the system to some known state.
Unrecoverable Error	The function detected some problem from which the system could not reestablish its state, with respect to the application, at the time when that call was issued. It is possible that the application cannot perform some corrective action to restore the system to some known state.

The WinGetLastError and WinGetErrorInfo functions can be used to find out more about an error (or warning) that occurs as a result of executing a call.

Chapter 2. Data Types

This chapter describes data types in a metalanguage (a form that is not language-dependent). When programming you should use the appropriate bindings book. For further information about metalanguage see "Notation Conventions" on page 1-1.

ACCEL Accelerator structure.

options *(BIT16)*
Options.

key *(USHORT)*
Key.

cmd *(USHORT)*
Command code.

The value to be placed in the **cmd** parameter of a WM_HELP, a WM_COMMAND or a WM_SYSCOMMAND.

ACCELTABLE Accelerator-table structure.

count *(COUNT2)*
Number of accelerator entries.

codepage *(USHORT)*
Code page for accelerator entries.

accel *(ACCEL****count)***
Accelerator entries.

The default accelerator table has the following 16 entries:

Options		Key	Command
HELP	VIRTUALKEY	VK_F1	0
SYSCOMMAND ALT	VIRTUALKEY	VK_F4	SC_CLOSE
SYSCOMMAND ALT	VIRTUALKEY	VK_ENTER	SC_RESTORE
SYSCOMMAND ALT	VIRTUALKEY	VK_NEWLINE	SC_RESTORE
SYSCOMMAND ALT	VIRTUALKEY	VK_F5	SC_RESTORE
SYSCOMMAND ALT	VIRTUALKEY	VK_F6	SC_NEXTFRAME
SYSCOMMAND ALT	VIRTUALKEY	VK_F7	SC_MOVE
SYSCOMMAND ALT	VIRTUALKEY	VK_F8	SC_SIZE
SYSCOMMAND ALT	VIRTUALKEY	VK_F9	SC_MINIMIZE
SYSCOMMAND ALT	VIRTUALKEY	VK_F10	SC_MAXIMIZE
SYSCOMMAND	VIRTUALKEY	VK_F10	SC_APPMENU
SYSCOMMAND LONEKEY	VIRTUALKEY	VK_ALT	SC_APPMENU
SYSCOMMAND LONEKEY	VIRTUALKEY	VK_ALTGRAF	SC_APPMENU
SYSCOMMAND ALT	VIRTUALKEY	VK_SPACE	SC_SYSMENU
SYSCOMMAND SHIFT	VIRTUALKEY	VK_ESC	SC_SYSMENU
SYSCOMMAND CONTROL	VIRTUALKEY	VK_ESC	SC_TASKMANAGER

ARCPARAM Arc-parameters structure.

p *(LONG)*
P coefficient.

q *(LONG)*
Q coefficient.

r *(LONG)*
R coefficient.

s *(LONG)*
S coefficient.

AREABUNDLE Area-attributes-bundle structure.

color *(LONG)*
Area foreground color.

backcolor *(LONG)*
Area background color.

mixmode *(USHORT)*
Area foreground-mix mode.

backmixmode *(USHORT)*
Area background-mix mode.

set *(USHORT)*
Pattern set.

symbol *(USHORT)*
Pattern symbol.

refpoint *(POINT)*
Pattern reference point.

ATOM Atom identity.

BANDRECT Rectangle structure, used for the coordinates of an output band (see DevEscape).

An empty rectangle is one for which **xleft** is greater than **xright**, or **ybottom** is greater than **ytop**.

xleft *(LONG)*
x coordinate of left edge of rectangle.

ybottom *(LONG)*
y coordinate of bottom edge of rectangle.

xright *(LONG)*
x coordinate of right edge of rectangle.

ytop *(LONG)*
y coordinate of top edge of rectangle.

BITMAPINFO Bit-map information structure.

Each bit plane logically contains (**bitmapwidth*** **bitmapheight*** **bitcount**) bits, although the actual length can be greater because of padding.

length *(COUNT4B)*
Length of fixed portion of structure.

12 Only valid value.

bitmapwidth *(USHORT)*
Bit-map width in pels.

bitmapheight *(USHORT)*
Bit-map height in pels.

planes *(COUNT2)*
Number of bit planes.

bitcount *(COUNT2)*
Number of bits per pel within a plane.

color *(RGB*1)*
Array of RGB values.

This is a packed array of 24-bit RGB values. If there are N bits per pel (N = **planes*** **bitcount**), the array contains 2**N RGB values. However, if N = 24 the bit map does not need the *color* array because the standard-format bit map, with 24 bits per pel, is assumed to contain RGB values.

BITMAPINFOHEADER Bit-map information header structure.

Each bit plane logically contains (**bitmapwidth* bitmapheight* bitcount**) bits, although the actual length can be greater because of padding.

length *(COUNT4B)*
Length of structure.

12 Only valid value.

bitmapwidth *(USHORT)*
Bit-map width in pels.

bitmapheight *(USHORT)*
Bit-map height in pels.

planes *(COUNT2)*
Number of bit planes.

bitcount *(COUNT2)*
Number of bits per pel within a plane.

BIT1 Defines one BOOL value.

BIT4 Defines four independent BOOL values.

BIT6 Defines six independent BOOL values.

BIT8 Defines eight independent BOOL values.

BIT16 Defines sixteen independent BOOL values.

BIT32 Defines thirty-two independent BOOL values.

BOOL Boolean.

The *BOOL* data type is defined such that any nonzero value indicates TRUE, and a zero value indicates FALSE.

The constants TRUE and FALSE have been defined as 1 and 0 respectively. The constant TRUE should never be used in an equality comparison with a *BOOL* as this may result in incorrect results.

When an application needs to pass a TRUE value on a function call, or as a return value from a window procedure, it is recommended that it passes the value TRUE, which is 1, and not any nonzero value.

BUFFER Buffer.

BUNDLE Bundle data area. It is overlaid by one of these attribute bundle structures:

AREABUNDLE
CHARBUNDLE
IMAGEBUNDLE
LINEBUNDLE
MARKERBUNDLE.

BYTE Byte.

CATCHBUF Saved execution environment buffer.

savearea *(ULONG*4)*
Save area.

CHAR Single-byte character.

CHARBUNDLE Character-attributes bundle structure.

color *(LONG)*
Character foreground color.

backcolor *(LONG)*
Character background color.

mixmode *(USHORT)*
Character foreground-mix mode.

backmixmode *(USHORT)*
Character background-mix mode.

set *(USHORT)*
Character set.

precision *(USHORT)*
Character precision.

cell *(SIZEROF)*
Character cell size.

angle *(POINT)*
Character angle.

shear *(POINT)*
Character shear.

direction *(USHORT)*
Character direction.

CLASSINFO Class-information structure.

style *(ULONG)*
Class-style flags.

windowproc *(WNDPROC)*
Window procedure.

extra *(USHORT)*
Number of additional window words.

COMPOSED A 4-byte value that is composed by the concatenation of values of other data types.

For example, it could be composed of two values, one of which has a data type of *SHORT* and the other a data type of *USHORT*.

COUNT2 Count in the range 0 through 65 535.

COUNT2B Count of bytes, in the range 0 through 65 535.

COUNT2CH Count of characters, in the range 0 through 65 535.

COUNT4 Count in the range 0 through 4 294 967 295.

COUNT4B Count of bytes, in the range 0 through 4 294 967 295.

CPID Code-page identity.

CREATEPARAMS Create parameters structure.

This structure can be used to format the application-defined data area used in the WinCreateDlg, the WinLoadDlg and the WinDlgBox calls in order to conform to SAA.

length *(COUNT2B)*
Length of structure.

type *(BIT16)*
Reserved.

NULL Reserved value.

data *(BYTE*1)*
Application defined data area.

CREATESTRUCT Create-window-data structure.

presparams *(PRESDATA)*
Presentation parameters.

ctldata *(CTLDATA)*
Control data.

windowidentity *(USHORT)*
Window identifier.

behind *(HWND)*
Window behind which the window is to be placed.

owner *(HWND)*
Window owner.

cy *(SHORT)*
Window height.

cx *(SHORT)*
Window width.

y *(SHORT)*
y coordinate of origin.

x *(SHORT)*
x coordinate of origin.

style *(BIT32)*
Window style.

text *(STRL)*
Window text.

classname *(STRL)*
Registered window class name.

parent *(HWND)*
Parent window handle.

CTLDATA

Class-specific control data, beginning with a value conforming to a *COUNT2B* data type, which specifies the overall length of the data.

The following are cases of this data type:

BTNCDATA	Button control data. See page 13-2.
ENTRYFDATA	Entry field control data. See page 14-2.
FRAMECDATA	Frame control data. See page 15-3.
MLECDATA	Multi-line entry field control data. See page 18-2.
SBCDATA	Scroll bar control data. See page 20-1.

CURSORINFO

Cursor-information structure.

hwnd *(HWND)*
Window handle.

x *(SHORT)*
x coordinate.

y *(SHORT)*
y coordinate.

cx *(SHORT)*
Cursor width.

cy *(SHORT)*
Cursor height.

options *(BIT16)*
Options.

rect *(RECTL)*
Cursor box.

DDEINIT Dynamic-data-exchange initiation structure.

length *(COUNT2B)*
Length of structure.

This must be set to the length of the *DDEINIT* structure.

appname *(STRL)*
Application name.

Pointer to name of the server application.

Application names must not contain slashes or backslashes. These characters are reserved for future use in network implementations.

topic *(STRL)*
Topic.

Pointer to name of the topic.

DDESTRUCT Dynamic-data-exchange control structure.

length *(COUNT4B)*
Total length.

This is the length of this structure plus the item name and data, which occur after the **data** parameter.

status *(BIT16)*
Status.

Status of the data exchange.

DDE_FACK	Positive acknowledgement
DDE_FBUSY	Application is busy
DDE_FNODATA	No data transfer for advise
DDE_FACKREQ	Acknowledgements are requested
DDE_FRESPONSE	Response to WM_DDE_REQUEST
DDE_FNOTPROCESSED	DDE message not understood
DDE_FAPPSTATUS	A 1-byte field of bits that are reserved for application-specific returns.

format *(USHORT)*
Data format.

One of the DDE data formats.

DDEFMT_TEXT	Text format.
Other	DDE format registered with the atom manager, using the system atom table. The predefined DDE formats are guaranteed not to conflict with the values returned by the atom manager.

itemname *(OFFSET2B)*
Offset to item.

This is the offset to the item name referred to in this message, from the start of this structure.

data *(OFFSET2B)*
Offset to beginning of data.

This is the offset to the data, from the start of this structure.

For compatibility reasons, this data should not contain embedded pointers. Offsets should be used instead.

DEVOPENDATA Open-device data area. The format of this area is as a *DEVOPENSTRUC* structure.

DEVOPENSTRUC Open-device data structure.

The same structure is applicable to both DevOpenDC and SplQmOpen calls.

address *(STRL)*
Logical address.

This is required for an OD_DIRECT device being opened with DevOpenDC; it is the logical device address, such as "LPT1" on OS/2. Some drivers may accept a file name for this parameter, or even a named pipe. A driver may restrict the logical address to certain names because special hardware is involved, for example a printer driver that uses shared memory to access the memory of a laser printer.

Where output is to be queued (either for an OD_QUEUED device opened with DevOpenDC or for SplQmOpen), this is the name of the queue for the output device, and must always be supplied if it is not available from **Token**.

drivername *(STRL)*
Driver name.

A string containing the name of the Presentation Manager device driver (for example, "IBM4201"). This information must always be supplied if it is not available from **Token**.

driverdata *(DRIVDATA)*
Driver data.

Data that is to be passed directly to the Presentation Manager device driver. Whether any of this is required depends upon the device driver.

datatype *(STRL)*
Data type.

For a OD_QUEUED or OD_DIRECT device, this parameter defines the type of data that is to be (or was) queued as follows:

PM_Q_STD	Standard format
PM_Q_RAW	Raw format.

Note that a device driver can define other data types.

With DevOpenDC, for both of the above device types the default is supplied by the device driver if **datatype** is not specified. For any other device type, **datatype** is ignored.

With SplQmOpen, this parameter must always be supplied if it is not available from **Token**.

comment *(STRL)*
Comment.

This is a natural language description of the file for queued output. For example, this can be displayed by the spooler to the end user and is optional.

queueprocname *(STRL)*
Queue-processor name.

This is the name of the queue processor for queued output, and is usually the default.

queueprocparams *(STRL)*
Queue-processor parameters.

This is a parameter string for the queue processor for queued output, and is optional.

spoolerparams *(STRL)*
Spooler parameters.

This is a parameter string for the spooler, for queued output, and is optional. It has the following options, which must be separated by one or more blanks:

FORM=f Specifies a form name 'f'. This must be a valid form name for the printer. If more than one form is needed for the job, all of the required form names are supplied, separated by commas, as FORM=aaaa,bbbb,cccc; the first one is the one that is to be used first. See *HCINFO*.

A form name can be enclosed in double quotes to permit form names to contain the characters ',' ';' or '='. For example,

```
FORM="A","A4 with heading","C,D"
```

calls for three forms: 'A', 'A4 with heading', and 'C,D'. If a double quote is part of the form name, it should be supplied twice.

If this option is not specified, 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 1 through 99, with 99 being the highest. If it is not specified, a default priority of 50 is used.

networkparams *(STRL)*
Network parameters.

This is a parameter string for the network program for queued output, and is optional. The format of the parameter string is keyword=value, and the following keyword is defined (additional ones can be defined by the network program):

USER=u Specifies the user identifier 'u'. If it is not specified, a null user identifier is used.

DLGTEMPLATE Dialog-template structure.

length *(COUNT2B)*
Length of template.

type *(USHORT)*
Template format type.

codepage *(USHORT)*
Code page.

itemoffset *(OFFSET2B)*
Offset to dialog items.

templatestatus *(BIT16)*
Template status.

itemfocus *(INDEX2)*
Index of item to receive focus initially.

presparamsoffsets *(COUNT2)*
Count of presentation-parameter offsets.

dlgti *(DLGTITEM*1)*
Start of dialog items.

DLGTITEM Dialog-item structure.

itemstatus *(BIT16)*
Status.

children *(COUNT2)*
Count of children to this dialog item.

classlen *(COUNT2CH)*
Length of class name.

If zero, **classname** contains the hexadecimal equivalent of a preregistered class name.

classname *(OFFSET2B)*
Offset to class name.

If **classlen** is nonzero, this is the offset to a null-terminated ASCII string that contains the class name. If **classlen** is zero, this is of the form 0xhhhh, where hhhh is the hexadecimal equivalent of the preregistered class name.

textlen *(COUNT2CH)*
Length of text.

text *(OFFSET2B)*
Offset to text.

style *(BIT32)*
Dialog item window style.

The high-order 16 bits are the standard WS_* style bits (see page 12-2). The low-order 16 bits are available for class-specific use.

x *(SHORT)*
x coordinate of origin of dialog-item window.

y *(SHORT)*
y coordinate of origin of dialog-item window.

width *(SHORT)*
Dialog-item window width.

height *(SHORT)*
Dialog-item window height.

id *(IDENTITY)*
Identity.

presparams *(OFFSET2B)*
Reserved.

ctldata *(OFFSET2B)*
Offset to control data.

DRIVDATA

Driver-data structure.

length *(LONG)*
Length.

The length of the structure.

version *(LONG)*
Version.

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

devicename *(CHAR*32)*
Device name.

A string in a 32-byte field, identifying the particular device (model number, and so on). Again, valid values are defined by Presentation Manager device drivers.

generaldata *(BYTE*1)*
General data.

Data as defined by the Presentation Manager device driver.

The data type of this field is defined by the Presentation Manager device driver. It does not contain pointers, as these are not necessarily valid when passed to the device driver.

ERRINFO

Error-information structure.

fixederrinfo *(COUNT2B)*
Length of fixed data to this structure.

error *(ERRORID)*
Error identity.

This is identical to the value returned by the WinGetLastError function.

detaillevel *(COUNT2)*
Number of levels of detail.

This is the number of entries in the array of words pointed to by the following field. One level of detail is provided.

messages *(OFFSET2B)*
Offset to the array of message offsets.

binarydata *(OFFSET2B)*
Offset to the binary data.

This can contain additional information relating to the error.

ERRORID Error identity.

FATTRS Font-attributes structure.

length *(LENGTH2)*
Length of record.

selection *(BIT16)*
Selection indicators.

Flags causing the following features to be simulated by the system.

Programming Note: If an italic flag is applied to a font that is itself defined as italic, the font is slanted further by italic simulation.

FATTR_SEL_ITALIC	Generate *italic* font.
FATTR_SEL_UNDERSCORE	Generate underscored font.
FATTR_SEL_BOLD	Generate **bold** font.
	Note: The resulting characters are wider than those in the original font.
FATTR_SEL_STRIKEOUT	Generate font with ~~overotruok~~ characters.
FATTR_SEL_OUTLINE	Use an outline font with hollow characters.
	If this flag is not set, outline font characters are filled. Setting this flag normally gives better performance, and for sufficiently small characters there may be little visual difference.

match *(PROPERTY4)*
Matched-font identity.

facename *(CHAR*FACESIZE)*
Typeface name.

The typeface name of the font, for example, Tms Rmn.

registry *(IDENTITY)*
Registry identifier.

Font registry identifier (zero if unknown).

codepage *(CPID)*
Code page.

maxbaselineext *(LONG)*
Maximum base-line extension.

For raster fonts, this should be the height of the required font, in world coordinates.

For outline fonts, this should be zero.

avecharwidth *(WIDTH4)*
Average character width.

For raster fonts, this should be the width of the required font, in world coordinates.

For outline fonts, this should be zero.

type *(BIT16)*
Type indicators.

FATTR_TYPE_KERNING	Enable kerning (PostScript only).
FATTR_TYPE_ANTIALIASED	Antialiased font required. Only valid if supported by the device driver.

fontuse *(BIT16)*
Font-use indicators.

These flags indicate how the font is to be used. They affect presentation speed and font quality.

FATTR_FONTUSE_NOMIX
Text is not mixed with graphics and may be written without regard to any interaction with graphics objects.
FATTR_FONTUSE_OUTLINE
Select an outline (vector) font. The font characters may be used as part of a path definition.
FATTR_FONTUSE_TRANSFORMABLE
Characters may be transformed (for example, scaled, rotated, or sheared).

FFDESCS Font-file descriptor.

familyname *(STR32)*
Family name.

facename *(STR32)*
Face name.

FIXED Signed-integer fraction (16:16). This can be treated as a *LONG* where the value has been multiplied by 65 536.

integer *(SHORT)*
Integral component.

fraction *(USHORT)*
Fractional component.

FONTMETRICS Font-metrics structure.

familyname *(CHAR*FACESIZE)*
Family name.

The family name of the font that describes the basic appearance of the font, for example, Roman.

facename *(CHAR*FACESIZE)*
Facename.

The typeface name that defines the particular font, for example, Tms Rmn Bold Italic.

registry *(IDENTITY)*
Registry identifier.

The IBM registered number (or zero).

codepage *(CPID)*
Code page.

Defines the registered code page supported by the font. For example, the original IBM PC code page is 437.

Where a font contains special symbols for which there is no registered code page, then code page 65400 should be used.

emheight *(LONG)*
Em height.

The nominal height above the baseline for uppercase characters in world coordinate units.

xheight *(LONG)*
x height.

The nominal height above the baseline for lowercase characters (ignoring ascenders) in world coordinate units.

maxascender *(LONG)*
Maximum ascender.

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

maxdescender *(LONG)*
Maximum descender.

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

lowercaseascent *(LONG)*
Lowercase ascent.

The maximum height above the baseline reached by any part of any lowercase (Latin 'a' through 'z') symbol in the font in world coordinate units.

lowercasedescent *(LONG)*
Lowercase descent.

The maximum depth below the baseline reached by any part of any lowercase (Latin 'a' through 'z') symbol in the font in world coordinate units.

internalleading *(LONG)*
Internal leading.

The amount of white space normally separating lines of text where lines are spaced by the maximum baseline extent (world coordinate units). This space may contain accents.

externalleading *(LONG)*
External leading.

The white space, in addition to the internal leading, to separate lines of text (world coordinate units). This value may be zero.

avecharwidth *(WIDTH4)*
Average character width.

Weighted average inter-character increment for the font in world coordinate units.

maxcharinc *(WIDTH4)*
Maximum character increment.

The maximum character increment for the font in world coordinate units.

eminc *(WIDTH4)*
Em increment.

A formatting measure determined by the font designer.

Programming Note: It is sometimes equal to the character increment of the 'M' graphic character and is returned in world coordinate units.

maxbaselineext *(WIDTH4)*
Maximum baseline extent.

This is the height of the character cell in world coordinate units.

The recommended line spacing can be computed by adding the **maxbaselineext** to the recommended **externalleading**.

charslope *(SHORT)*
Character slope.

Defines the nominal slope for the characters of a font. The slope is defined in degrees increasing clockwise from the vertical. An *Italic* font is an example of a font with a nonzero slope.

Programming Note: This angle is contained in a two-part unsigned number. The first 9 bits contain a value in the range 0 through 359, representing the number of degrees in the rotation. The next 6 bits contain a number in the range 0 through 59, representing the number of minutes in the rotation. The final bit is reserved 0. Values outside the specified ranges are not valid.

inlinedir *(SHORT)*
Inline direction.

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

Programming Note: This angle is contained in a two-part unsigned number. The first 9 bits contain a value in the range 0 through 359, representing the number of degrees in the rotation. The next 6 bits contain a number in the range 0 through 59, representing the number of minutes in the rotation. The final bit is reserved 0. Values outside the specified ranges are not valid.

charrot *(SHORT)*
Character rotation.

The rotation of the character glyphs with respect to the baseline.

Programming Note: This angle is contained in a two-part unsigned number. The first 9 bits contain a value in the range 0 through 359, representing the number of degrees in the rotation. The next 6 bits contain a number in the range 0 through 59, representing the number of minutes in the rotation. The final bit is reserved 0. Values outside the specified ranges are not valid.

weightclass *(PROPERTY2)*
Weight class.

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

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

widthclass *(PROPERTY2)*
Width class.

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

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

xdeviceres *(SHORT)*
x device resolution.

The x-resolution of the target device. For raster fonts the units are pels/inch and for outline fonts the units are the logical units defining the drawing grid.

ydeviceres *(SHORT)*
y device resolution.

The y-resolution of the target device. For raster fonts the units are pels/inch and for outline fonts the units are the logical units defining the drawing grid.

firstchar *(SHORT)*
First character.

The code point of the first character in the font.

lastchar *(SHORT)*
Last character.

The code point of the last character in the font, expressed as an offset from **firstchar**.

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

defaultchar *(SHORT)*
Default character.

The code point that is used if a code point outside the range supported by the font is used, expressed as an offset from **firstchar**.

breakchar *(SHORT)*
Break character.

The code point that represents the 'space' or 'break' character for this font, expressed as an offset from **firstchar**. For example, if the first character is the space in code page 850, **firstchar** = 32, and **breakchar** = 0.

nominalpointsize *(SHORT)*
Nominal point size.

The height of the font specified in *decipoints* (1/720 inch). The nominal size is the size for which the font is designed.

minimumpointsize *(SHORT)*
Minimum point size.

The minimum height in decipoints to which the font can be scaled down for display while retaining fidelity.

maximumpointsize *(SHORT)*
Maximum point size.

The maximum height in decipoints to which the font can be scaled up for display while retaining fidelity.

type *(BIT16)*
Type indicators.

Contains this information:

FMETRICS_FIXED
Characters in the font have the same fixed width.

FMETRICS_LICENSED
Licensed (protected) font.

FMETRICS_KERNED
Font contain kerning information.

FMETRICS_ANTI_ALIASED
Font contains more than one plane.

FMETRICS_64K
Font is larger than 64K bytes in size.

If the following two bits are false than the font is for single-byte code pages. One of the bits may be set.

FMETRICS_DOUBLE_BYTE
Font for double-byte code pages.

FMETRICS_MIXED_BYTE
Font for mixed single/double-byte code pages.

defn *(BIT16)*
Definition indicators.

Contains the following font definition data:

FMETRICS_OUTLINE
Font is a vector (outline) font, otherwise it is a bit-map font.

FMETRICS_GENERIC
Font is in a format that can be used by the GPI, otherwise it is a device font.

selection *(BIT16)*
Selection indicators.

Contains information about the font patterns in the physical font.

Note: The flags do not reflect simulations applied to the physical font.

FMETRICS_ITALIC
Font contains Italic characters otherwise they are upright.

FMETRICS_UNDERSCORE
Characters are underscored.

FMETRICS_NEGATIVE
Characters have their foreground and background reversed.

FMETRICS_OUTLINED
Outline (hollow) characters, otherwise they are solid.

FMETRICS_OVERSTRUCK
Characters are overstruck.

capabilities *(BIT16)*
Capabilities.

This tells the engine the kind of operations it can perform on the font.

FMETRICS_NOMIX
Characters may be mixed with graphics.

subscriptxsize *(LONG)*
Subscript x size.

The recommended horizontal size for subscripts for this font in world coordinate units.

subscriptysize *(LONG)*
Subscript y size.

The recommended vertical size for subscripts for this font in world coordinate units.

subscriptxoffset *(LONG)*
Subscript x offset.

The recommended baseline x-offset for subscripts for this font in world coordinate units.

subscriptyoffset *(LONG)*
Subscript y offset.

The recommended baseline y-offset for subscripts for this font in world coordinate units.

superscriptxsize *(LONG)*
Superscript x size.

The recommended horizontal size for superscripts for this font in world coordinate units.

superscriptysize *(LONG)*
Superscript y size.

The recommended vertical point size for superscripts for this font in world coordinate units.

superscriptxoffset *(LONG)*
Superscript x offset.

The recommended baseline x-offset for superscripts for this font in world coordinate units.

superscriptyoffset *(LONG)*
Superscript y offset.

The recommended baseline y-offset for superscripts for this font in world coordinate units.

underscoresize *(LONG)*
Underscore size.

The width of the underscore stroke in world coordinate units.

underscoreposition *(LONG)*
Underscore position.

The position of the underscore stroke from the baseline in world coordinate units.

strikeoutsize *(LONG)*
Strikeout size.

The width of the strikeout stroke in world coordinate units.

strikeoutposition *(LONG)*
Strikeout position.

The position of the strikeout stroke relative to the baseline in world coordinate units.

kerningpairs *(SHORT)*
Kerning pairs.

The number of kerning pairs in the kerning pair table.

familyclass *(SHORT)*
Font family design classification.

This value contains a font class and its subclass.

match *(PROPERTY4)*
Matched font identity.

This uniquely identifies the font to a device driver. A positive match number signifies that the font is a generic (engine) font while a negative number indicates a device font (a native or downloadable font).

GRADIENT Direction-vector structure.

ax *(LONG)*
x component of direction.

ay *(LONG)*
y component of direction.

GRADIENTL Direction-vector structure.

x *(LONG)*
x component of direction.

y *(LONG)*
y component of direction.

HAB Anchor-block handle.

HACCEL Accelerator-table handle.

HANDLE The handle of a resource.

HAPP Handle of an application started by the WinInstStartApp function.

HATOMTBL Atom-table handle.

HBITMAP Bit-map handle.

HCINFO Hardcopy-capabilities structure.

formname *(CHAR*32)*
Form name.

xwidth *(PROPERTY4)*
Width (left-to-right) in millimeters.

yheight *(PROPERTY4)*
Height (top-to-bottom) in millimeters.

xleftclip *(PROPERTY4)*
Left clip limit in millimeters.

ybottomclip *(PROPERTY4)*
Bottom clip limit in millimeters.

xrightclip *(PROPERTY4)*
Right clip limit in millimeters.

ytopclip *(PROPERTY4)*
Top clip limit in millimeters.

xpels *(PROPERTY4)*
Number of pels between left and right clip limits.

ypels *(PROPERTY4)*
Number of pels between bottom and top clip limits.

attributes *(LONG)*
Attributes of the form identifier.

HCAPS_CURRENT Currently installed form.

HCAPS_SELECTABLE Form is available from an alternate form source, without operator intervention.

The value returned is the sum of the applicable values. The bits in the field that are affected by each piece of information are separate.

HDC Device-context handle.

HELPINIT Help manager initialization structure.

length *(COUNT2B)*
Count of bytes of the initialization structure.

returncode *(ULONG)*
Value returned by the help manager from initialization.

0 Initialization was successful.

tutorialname *(STRL)*
Indicates to the help manager that the application has a tutorial program.

NULL The application either does not have a tutorial program, or the tutorial name is specified in each help panel definition.
Other Default tutorial name.

helptable *(HELPTABLE)*
Help table.

The help table or the identity of the help table. If this is the identity of the help table in a resource file, then the low-order word contains the identity of the table and the high-order word must be X'FFFF'.

The help table associates each application window with its help subtable and the identity of its extended help panel.

helptablemodule *(RESID)*
Resource file identity.

If the **HelpTable** contains the identity of the help table, this field identifies the module handle returned by the DosLoadModule call by which the application loaded the resource file.

NULL The resource file containing the help table was appended to the application's .EXE file.
Other Resource file identity.

accelactionbarmodule *(HMODULE)*
Handle of the containing DLL.

The handle of the DLL which contains the accelerator table and action bar template to be used by the help manager.

NULL The resource file containing the tailored accelerator table and action bar was appended to the application's .EXE file.
Other Handle of the DLL.

acceltable *(IDENTITY)*
Identity of the accelerator table.

The accelerator table resides in the DLL provided in the **accelactionbarmodule** field.

NULL Use the default accelerator table
Other Identity of the accelerator table.

actionbar *(IDENTITY)*
Identity of the action bar template used by the help manager.

The action bar template resides in the DLL provided in the **accelactionbarmodule** field.

NULL Use the default action bar
Other Identity of the action bar.

helpwindowtitle *(STRL)*
Window title for each help window.

showpanelid *(BIT16)*
Show panel identity indicator.

The constants corresponding to the panel identity flags are in the PMHELP.H include file.

CMIC_SHOW_PANEL_ID Show the panel identity on a help panel
CMIC_HIDE_PANEL_ID Do not show the panel identity on a help panel.

helplibraryname *(STRL)*
Names of the help panel libraries that the help manager searches on each help request.

The names must be separated by a blank.

The help manager looks for the libraries in the path set by the HELP environment variable. If the variable is not set, then the help manager will look for the libraries in the default directory.

HELPSUBTABLE Help subtable structure.

subitemsize *(USHORT)*
Number of words in each entry of the help subtable.

This parameter defines the size of an entry in the help subtable. An entry consists of a **field**, parameter, a **helppanel** parameter and an array of optional integers in the **optionalintegers** parameter.

2 The minimum number of words in each entry of the help subtable
Other Number of words in each entry of the help subtable.

helpsubtableitems *(HELPSUBTABLEITEM*)*
Array of help subtable entries.

The last entry of the array has its **field** parameter set to NULL.

HELPSUBTABLEITEM Help subtable entry structure.

field *(IDENTITY)*
Identity of the field from which the user can request help.

The identity can be a control identity, menu item identity, or a message box identity, and must be unique across the table.

NULL Last help subtable entry
X'ffff' Reserved for use by the help manager
Other Field identity.

helppanel *(IDENTITY)*
Identity of the contextual help panel.

optionalintegers *(USHORT***subitemsize***-2)*
Application-related integers.

Used to store information that the help manager is to ignore.

HELPTABLE Help table structure.

This is an array of help table entries, each of which has the structure defined below, the last entry of the array having its **extpanel** parameter set to NULL.

appwindow *(IDENTITY)*
Application window identity.

helpsubtable *(HELPSUBTABLE)*
Help subtable for this application window.

extpanel *(IDENTITY)*
Identity of the extended help panel for the application window.

NULL Last help table entry
Other Extended help panel identity.

HENUM Window-enumeration handle.

HHEAP Heap handle.

HINI Initialization-file handle.

HLIB Library handle.

HMF Metafile handle.

HMODULE Module handle

HMQ Message-queue handle.

HPOINTER Pointer handle.

HPROGARRAY Array of program handles.

ahprog *(HPROGRAM*1)*
Program handle array.

HPROGRAM Program handle.

HPS Presentation-space handle.

HRGN Region handle.

HSEM Semaphore handle.

HSPL Spooler handle.

HSWITCH Switch-list-entry handle.

HVPS VIO presentation-space handle.

HWND Window handle.

IDENTITY Identity value. Up to 65 536 different identities are available.

IDENTITY4 Identity value. Up to 4 294 967 296 different identities are available.

IMAGEBUNDLE Image-attributes bundle structure.

color *(LONG)*
Image foreground color.

backcolor *(LONG)*
Image background color.

mixmode *(USHORT)*
Image foreground-mix mode.

backmixmode *(USHORT)*
Image background-mix mode.

INDEX2 Index value, in the range 0 through 65 535.

IPT Insertion point for multi-line entry field.

KERNINGPAIRS Kerning-pair records structure.

firstchar *(SHORT)*
First character of pair.

secondchar *(SHORT)*
Second character of pair.

kerningamount *(SHORT)*
Amount of kerning for this pair.

LENGTH1 Length value, in the range 0 through 255.

LENGTH2 Length value, in the range 0 through 65 535.

LENGTH4 Length value, in the range 0 through 4 294 967 295.

LHANDLE The handle of a resource.

LINEBUNDLE Line-attributes bundle structure.

color *(LONG)*
Line foreground color.

backcolor *(LONG)*
Reserved.

mixmode *(USHORT)*
Line foreground-mix mode.

backmixmode *(USHORT)*
Reserved.

width *(ROF)*
Line width.

geomwidth *(ROL)*
Geometric line width.

type *(USHORT)*
Line type.

end *(USHORT)*
Line end.

join *(USHORT)*
Line join.

LONG Signed integer in the range −2 147 483 648 through 2 147 483 647.

Note: Where this data type represents a graphic coordinate in world or model space, its value is restricted to −134 217 728 through 134 217 727. A graphic coordinate in device or screen coordinates, is restricted to −32 768 through 32 767. Its value may be further restricted by any transforms currently in force, including the positioning of the origin of the window on the screen. In particular, coordinates in world or model space must not generate coordinate values after transformation (that is, in device or screen space) outside the range −32 768 through 32 767.

MARGSTRUCT Multi-line entry field margin information

flags *(BIT16)*

This gives the margin in which the event occurred.

The left and right margins are defined as including the corners at the top and bottom. The limits of the top and bottom margins are between the left and right margins.

MLF_MARGINLEFT
MLF_MARGINRIGHT
MLF_MARGINTOP
MLF_MARGINBOTTOM

moumsg *(USHORT)*

The message identity of the original mouse event.

near *(IPT)*

The insertion point nearest to the margin event.

MARKERBUNDLE Marker-attributes bundle structure.

color *(LONG)*
Marker foreground color.

backcolor *(LONG)*
Marker background color.

mixmode *(USHORT)*
Marker foreground-mix mode.

backmixmode *(USHORT)*
Marker background-mix mode.

set *(USHORT)*
Marker set.

symbol *(USHORT)*
Marker symbol.

cell *(SIZEROF)*
Marker cell.

MATRIX
Matrix-elements structure.

m11 *(FIXED)*
First element of first row.

m12 *(FIXED)*
Second element of first row.

m13 *(LONG)*
Third element of first row.

m21 *(FIXED)*
First element of second row.

m22 *(FIXED)*
Second element of second row.

m23 *(LONG)*
Third element of second row.

m31 *(LONG)*
First element of third row.

m32 *(LONG)*
Second element of third row.

m33 *(LONG)*
Third element of third row.

MATRIXLF
Matrix-elements structure.

m11 *(FIXED)*
First element of first row.

m12 *(FIXED)*
Second element of first row.

m13 *(LONG)*
Third element of first row.

m21 *(FIXED)*
First element of second row.

m22 *(FIXED)*
Second element of second row.

m23 *(LONG)*
Third element of second row.

m31 *(LONG)*
First element of third row.

m32 *(LONG)*
Second element of third row.

m33 *(LONG)*
Third element of third row.

MENUITEM Menu item.

position *(SHORT)*
Position.

style *(BIT16)*
Style.

attribute *(BIT16)*
Attribute.

identity *(IDENTITY)*
Identity.

submenu *(HWND)*
Submenu.

item *(ULONG)*
Item.

MLE_SEARCHDATA Search structure for multi-line entry field.

structsize *(SHORT)*
Size of *MLE_SEARCHDATA* structure.

findstringlength *(SHORT)*
Length of **findstring** string.

findstring *(STRL)*
String to search for.

start *(IPT)*
Point at which to start search, or point where string was found.

non-negative	Point at which to start search.
negative	Start search from current cursor location.

stop *(IPT)*
Point at which to stop search.

non-negative	Point at which to stop search.
negative	Stop search at end of text.

foundstringlength *(SHORT)*
Length of string found at **start**.

changestringlength *(SHORT)*
Length of replacement string.

changestring *(STRL)*
Replacement string.

MPARAM 4-byte message-dependent parameter structure.

Certain elements of information, placed into the parameters of a message, have data types that do not use all 4 bytes of this data type. The rules governing these cases are:

BOOL	The value is contained in the low word and the high word is zero.
SHORT	The value is contained in the low word and its sign is extended into the high word.
USHORT	The value is contained in the low word and the high word is zero.
NULL	The entire 4 bytes are zero.

The structure of this data type depends on the message. For details, see the description of the particular message.

MRESULT 4-byte message-dependent reply parameter structure.

Certain elements of information, placed into the parameters of a message, have data types that do not use all 4 bytes of this data type. The rules governing these cases are:

BOOL The value is contained in the low word and the high word is zero.

SHORT The value is contained in the low word and its sign is extended into the high word.

USHORT The value is contained in the low word and the high word is zero.

NULL The entire 4 bytes are zero.

The structure of this data type depends on the message. For details, see the description of the particular message.

MQINFO Message-queue information structure.

b *(COUNT2)*
Length of structure.

pid *(PID)*
Process identity.

tid *(TID)*
Thread identity.

msgs *(COUNT2)*
Message count.

reserved *(STORAGE)*
Reserved.

MT Menu template.

length *(COUNT2B)*
Length of menu template in bytes.

version *(USHORT)*
Template version.

codepage *(USHORT)*
Code page.

itemoffset *(USHORT)*
Offset of item from start of template.

count *(COUNT2)*
Count of menu items.

reserved *(USHORT)*
Reserved.

menuitems *(MTI*count)*
Menu items.

MTI Menu template item.

style *(BIT16)*
Style.

attribute *(BIT16)*
Attributes.

item *(IDENTITY)*
Item identity.

c *(CHAR*2)*
Item data.

The format and length of this depend upon the value of **style**.

OFFSET2B Offset value in bytes, in the range 0 through 65 535.

OVERFLOW Overflow error structure for multi-line entry field.

errind *(ULONG)*
One or more EFR_* flags.

bytesover *(LONG)*
Number of bytes over the limit.

horzover *(PIX)*
Number of pels over the horizontal limit.

vertover *(PIX)*
Number of pels over the vertical limit.

OWNERITEM Owner item.

hwnd *(HWND)*
Window handle.

hps *(HPS)*
Presentation-space handle.

state *(BIT16)*
State.

attribute *(BIT16)*
Attribute.

stateold *(BIT16)*
Old state.

attributeold *(BIT16)*
Old attribute.

itemrectangle *(RECTL)*
Item rectangle.

identity *(SHORT)*
Item identity.

item *(ULONG)*
Item.

PARAM Presentation parameter attribute definition.

attrtype *(IDENTITY4)*
Attribute type identity.

These identities are in the range of X'00000000' to X'FFFFFFFF'. The window manager uses values of this parameter in the range X'00000000' to X'0000BFFF', therefore an application should not define private presentation parameter attribute identities in this range. An application should use the WinAddAtom call to guarantee obtaining a unique identity.

PP_FOREGROUNDCOLOR	Foreground color (in RGB) attribute.
PP_BACKGROUNDCOLOR	Background color (in RGB) attribute.
PP_FOREGROUNDCOLORINDEX	Foreground color index attribute.
PP_BACKGROUNDCOLORINDEX	Background color index attribute.
PP_HILITEFOREGROUNDCOLOR	Highlighted foreground color (in RGB) attribute, for example for selected menu items.
PP_HILITEBACKGROUNDCOLOR	Highlighted background color (in RGB) attribute.
PP_HILITEFOREGROUNDCOLORINDEX	Highlighted foreground color index attribute.
PP_HILITEBACKGROUNDCOLORINDEX	Highlighted background color index attribute.
PP_DISABLEDFOREGROUNDCOLOR	Disabled foreground color (in RGB) attribute.

PP_DISABLEDBACKGROUNDCOLOR	Disabled background color (in RGB) attribute.
PP_DISABLEDFOREGROUNDCOLORINDEX	Disabled foreground color index attribute.
PP_DISABLEDBACKGROUNDCOLORINDEX	Disabled background color index attribute.
PP_BORDERCOLOR	Border color (in RGB) attribute.
PP_BORDERCOLORINDEX	Border color index attribute.
PP_FONTNAMESIZE	Font name and size attribute.

attrvaluelen *(COUNT4B)*
Byte count of the **attrvalue** parameter.

attrvalue *(BYTE*1)*
Attribute value.

The format of a value depends on the attribute type identity as follows:

PP_FOREGROUNDCOLOR	Foreground color value of data type *RGB*.
PP_BACKGROUNDCOLOR	Background color value of data type *RGB*.
PP_FOREGROUNDCOLORINDEX	Foreground color index value of data type *LONG*.
PP_BACKGROUNDCOLORINDEX	Background color index value of data type *LONG*.
PP_HILITEFOREGROUNDCOLOR	Highlighted foreground color value of data type *RGB*.
PP_HILITEBACKGROUNDCOLOR	Highlighted background color value of data type *RGB*.
PP_HILITEFOREGROUNDCOLORINDEX	Highlighted foreground color index value of data type *LONG*.
PP_HILITEBACKGROUNDCOLORINDEX	Highlighted background color index value of data type *LONG*.
PP_DISABLEDFOREGROUNDCOLOR	Disabled foreground color value of data type *RGB*.
PP_DISABLEDBACKGROUNDCOLOR	Disabled background color value of data type *RGB*.
PP_DISABLEDFOREGROUNDCOLORINDEX	Disabled foreground color index value of data type *LONG*.
PP_DISABLEDBACKGROUNDCOLORINDEX	Disabled background color index value of data type *LONG*.
PP_BORDERCOLOR	Border color value of data type *RGB*.
PP_BORDERCOLORINDEX	Border color index value of data type *LONG*.
PP_FONTNAMESIZE	Font name and size value of data type *STRL*. The string is in two parts, separated by a period. The first part is the font point size and the second part is the font facename, for example, "12.Helv".

PIBSTRUCT Program-information-block structure.

progt *(PROGTYPE)*
Program type and visibility.

title *(CHAR*MAXNAMEL+1)*
Program title (null-terminated).

iconfilename *(CHAR*MAXPATHL+1)*
Program icon filename (null-terminated).

executable *(CHAR*MAXPATHL+1)*
Executable file name (null-terminated).

startupdir *(CHAR*MAXPATHL+1)*
Start-up directory (null-terminated).

initial *(XYWINSIZE)*
Initial window position and size.

res1 *(COUNT2CH)*
Reserved; must be zero.

res2 *(HANDLE)*
Reserved; must be zero.

environmentvarslen *(COUNT2CH)*
Environment string length.

environmentvars *(STR)*
Environment string.

programparameterlen *(COUNT2CH)*
Parameter string length.

programparameter *(STR)*
Parameter string.

PID Process identity.

PIX Pel count for multi-line entry field.

POINT Point structure.

x *(LONG)*
x coordinate.

y *(LONG)*
y coordinate.

POINTERINFO Pointer-information structure.

flag *(BOOL)*
Bit-map size indicator.

TRUE	Pointer-sized bit map
FALSE	Icon-sized bit map.

xhotspot *(SHORT)*
x coordinate of action point.

yhotspot *(SHORT)*
y coordinate of action point.

hbitmap *(HBITMAP)*
Bit-map handle of pointer.

color *(HBITMAP)*
Bit-map handle of color bit map.

POINTL Point structure.

x *(LONG)*
x coordinate.

y *(LONG)*
y coordinate.

POINTS Point structure (short integer).

x *(SHORT)*
x coordinate.

y *(SHORT)*
y coordinate.

PRESPARAMS Presentation parameter data.

attrcount *(COUNT4B)*
Byte count of the **param** parameter.

param *(PARAM*1)*
Array of attribute parameters.

PRESDATA Class-specific control presentation data, conforming to a *PRESPARAM* data type.

PRFPROFILE Profile structure.

userlen *(COUNT4)*
Length of user profile name.

username *(STRL)*
User profile name.

syslen *(COUNT4)*
Length of system profile name.

sysname *(STRL)*
System profile name.

PROC Procedure identifier.

PROGCATEGORY Program category.

PROGDETAILS Program details structure.

length *(COUNT4B)*
Length of structure.

type *(PROGTYPE)*
Program type.

pad1 *(BIT16*3)*
Reserved.

title *(STRL)*
Title.

executable *(STRL)*
Executable file name.

parameters *(STRL)*
Parameter string.

startupdir *(STRL)*
Start-up directory.

icon *(STRL)*
Icon-file name.

environment *(STRLLIST)*
Environment string. (List of null-terminated strings, ending with an extra null.)

wininitial *(SWP)*
Initial window position and size.

pad2 *(BIT16*5)*
Reserved.

PROGRAMENTRY Program-title structure.

hprog *(HPROGRAM)*
Program handle.

programtype *(PROGTYPE)*
Program type.

programtitle *(CHAR*MAXNAMEL+1)*
Program title (null-terminated).

PROGTITLE Program-title structure.

hprog *(HPROGRAM)*
Program handle.

type *(PROGTYPE)*
Program type.

pad *(BIT16*3)*
Reserved.

title *(STRL)*
Program title.

PROGTYPE Program-type structure.

progc *(PROGCATEGORY)*
Program category:

PROG_DEFAULT	Default application
PROG_FULLSCREEN	Full-screen application
PROG_PM	Presentation Manager application
PROG_REAL	PC DOS executable process
PROG_WINDOWABLEVIO	Text-windowed application.

visible *(BIT8)*
Visibility attribute.

When testing this field, allow for the possibility that other bits may be defined in the future. SHE_INVISIBLE and SHE_PROTECTED can be used to mask the visibility and protected flags, respectively.

SHE_VISIBLE	Visible
SHE_INVISIBLE	Invisible
SHE_UNPROTECTED	Unprotected
SHE_PROTECTED	Protected.

PROPERTY2 Property value. Up to 65 536 different properties are available.

PROPERTY4 Property value. Up to 4 294 967 296 different properties are available.

PSZ Pointer to a null-terminated string.

PVOID Pointer to a data type of undefined format.

QMOPENDATA Open queue-manager data area. The format of this area is as a *DEVOPENSTRUC* structure.

QMSG Message structure.

hwnd *(HWND)*
Window handle.

msgid *(USHORT)*
Message identity.

param1 *(MPARAM)*
Parameter 1.

param2 *(MPARAM)*
Parameter 2.

time *(ULONG)*
Message time.

point *(POINTL)*
Pointer position when message was generated.

RECT

Rectangle structure.

Unless otherwise stated, points on the right-hand or top boundaries of the rectangle are not considered to be included in the rectangle; points on the left-hand or bottom boundaries (that are not also on the right-hand or top boundaries) are included in the rectangle.

xleft *(LONG)*
x coordinate of left edge of rectangle.

ybottom *(LONG)*
y coordinate of bottom edge of rectangle.

xright *(LONG)*
x coordinate of right edge of rectangle.

ytop *(LONG)*
y coordinate of top edge of rectangle.

RECTL

Rectangle structure.

xleft *(LONG)*
x coordinate of left-hand edge of rectangle.

ybottom *(LONG)*
y coordinate of bottom edge of rectangle.

xright *(LONG)*
x coordinate of right-hand edge of rectangle.

ytop *(LONG)*
y coordinate of top edge of rectangle.

RESID

Resource identity.

RGB

RGB color value.

blue *(BYTE)*
Blue component of the color definition.

green *(BYTE)*
Green component of the color definition.

red *(BYTE)*
Red component of the color definition.

RGNRECT

Region-rectangle structure.

start *(USHORT)*
Rectangle number from which to start enumerating.

Numbering starts from one.

count *(USHORT)*
Number of rectangles that can be returned.

Must be at least one.

retcount *(USHORT)*
Number of rectangles returned.

A value of less than **count** indicates that there are no more rectangles to enumerate.

direction *(USHORT)*
Direction in which the returned rectangles are to be ordered.

This ordering uses the leading edge of a rectangle.

RECTDIR_LFRT_TOPBOT	Left-to-right, top-to-bottom
RECTDIR_RTLF_TOPBOT	Right-to-left, top-to-bottom
RECTDIR_LFRT_BOTTOP	Left-to-right, bottom-to-top
RECTDIR_RTLF_BOTTOP	Right-to-left, bottom-to-top.

ROF Number representation, equivalent to *FIXED*.

ROL Number representation, equivalent to *LONG*.

SEGOFF 2-byte segment offset in bytes.

SFACTORS Scaling factors, see DevEscape.

x *(LONG)*
x scaling factor, as an exponent of 2.

y *(LONG)*
y scaling factor, as an exponent of 2.

SHORT Signed integer in the range –32 768 through 32 767.

SIZEF Size structure.

width *(FIXED)*
Width.

height *(FIXED)*
Height.

SIZEROF Size structure.

width *(FIXED)*
Width.

height *(FIXED)*
Height.

SIZEROL Size structure.

width *(LONG)*
Width.

height *(LONG)*
Height.

SMHSTRUCT Send-message-hook structure.

param2 *(MPARAM)*
Parameter 2.

param1 *(MPARAM)*
Parameter 1.

msgid *(USHORT)*
Message identity.

hwnd *(HWND)*
Window handle.

STORAGE Storage that may contain different data types.

STR String with an explicit length count.

STRCOND String with a length count that is determined by special values of its associated length parameter.

For example, positive values of the associated length parameter provide the length count explicitly, whereas the value –1 implies that the string is null-terminated and hence its length is implicit.

Where this data type occurs, the associated length parameter specifies the meanings of its values in determining the length count of the string.

STRL

Null-terminated string, that is, its length can be implied by the system.

Programming Note: Certain languages require the length of the string to be specified explicitly. See the definition of this data type in the appropriate language binding.

STRLLIST

A list of null-terminated strings (that is, a series of concatenated strings each of which has a *STRL* data type) terminated by an additional null.

STR8

String of 8 characters.

STR16

String of characters, with an implicit length, in a 16-byte field.

STR32

String of characters, with an implicit length, in a 32-byte field.

STR64

String of characters, with an implicit length, in a 64-byte field.

SWBLOCK

Switch-list block structure.

count *(COUNT2)*
Count of switch list entries.

swentry *(SWENTRY****count)***
Switch list entries.

SWCNTRL

Task-list control block structure.

hwnd *(HWND)*
Window handle.

icon *(HWND)*
Window-handle icon.

hprog *(HPROGRAM)*
Program handle.

process *(IDENTITY)*
Process identity.

session *(IDENTITY)*
Session identity.

visibility *(UCHAR)*
Visibility:

SWL_VISIBLE	Visible in startup list
SWL_INVISIBLE	Invisible in startup list
SWL_GRAYED	Item cannot be switched to (note that it is not actually grayed in the list).

jump *(BIT8)*
Jump indicator:

SWL_JUMPABLE	Participates in jump sequence
SWL_NOTJUMPABLE	Does not participate in jump sequence.

swtitle *(CHAR***MAXNAMEL+1)*
Switch-list control block title (null-terminated).

reserved *(BYTE)*
Reserved.

SWENTRY

Switch-list entry structure.

hswitch *(HSWITCH)*
Switch list entry handle.

swcntrl *(SWCNTRL)*
Switch list control block structure.

SWP Set-window-position structure.

options *(BIT16)*
Options.

In alphabetic order:

SWP_ACTIVATE
SWP_DEACTIVATE
SWP_HIDE
SWP_MAXIMIZE
SWP_MINIMIZE
SWP_MOVE
SWP_NOADJUST
SWP_NOREDRAW
SWP_RESTORE
SWP_SHOW
SWP_SIZE
SWP_ZORDER

height *(SHORT)*
Window height.

width *(SHORT)*
Window width.

y *(SHORT)*
y coordinate of origin.

x *(SHORT)*
x coordinate of origin.

behind *(HWND)*
Window behind which this window is placed.

hwnd *(HWND)*
Window handle.

TID Thread identity.

TIME Time interval in milliseconds.

TRACKINFO Tracking-information structure.

borderwidth *(SHORT)*
Border width.

The width of the left and right tracking sides.

borderheight *(SHORT)*
Border height.

The height of the top and bottom tracking sides.

gridwidth *(SHORT)*
Grid width.

The horizontal bounds of the tracking movements.

gridheight *(SHORT)*
Grid height.

The vertical bounds of the tracking movements.

xkeyboard *(SHORT)*
Character cell width movement for arrow key.

ykeyboard *(SHORT)*
Character cell height movement for arrow key.

track *(RECTL)*
Starting tracking rectangle.

This is modified as the rectangle is tracked and holds the new tracking position, when tracking is complete.

boundary *(RECTL)*
Boundary rectangle.

This is an absolute bounding rectangle that the tracking rectangle cannot extend; see also TF_ALLINBOUNDARY.

mintracksize *(POINTL)*
Minimum tracking size.

maxtracksize *(POINTL)*
Maximum tracking size.

options *(BIT16)*
Tracking options.

In alphabetic order:

TF_ALLINBOUNDARY The default tracking is such that some part of the tracking rectangle is within the bounding rectangle defined by **boundary**. This minimum size is defined by **borderwidth** and **borderheight**.

If TF_ALLINBOUNDARY is specified, the tracking is performed so that no part of the tracking rectangle ever falls outside of the bounding rectangle.

TF_BOTTOM Track the bottom side of the rectangle.

TF_GRID Tracking is restricted to the grid defined by **gridwidth** and **gridheight**.

TF_LEFT Track the left side of the rectangle.

TF_MOVE Track all sides of the rectangle.

TF_RIGHT Track the right side of the rectangle.

TF_SETPOINTERPOS The pointer is repositioned according to other flags as follows:

none	Pointer is centered in the tracking rectangle.
TF_MOVE	Pointer is centered in the tracking rectangle.
TF_LEFT	Pointer is vertically centered at the left of the tracking rectangle.
TF_TOP	Pointer is horizontally centered at the top of the tracking rectangle.
TF_RIGHT	Pointer is vertically centered at the right of the tracking rectangle.
TF_BOTTOM	Pointer is horizontally centered at the bottom of the tracking rectangle.

TF_STANDARD **width**, **height**, **gridwidth**, and **gridheight** are all multiples of **borderwidth** and **borderheight**.

TF_TOP Track the top side of the rectangle.

UCHAR Unsigned integer in the range 0 through 255.

ULONG
Unsigned integer in the range 0 through 4 294 967 295.

USERBUTTON
User-button data structure.

hwnd *(HWND)*
Window handle.

hps *(HPS)*
Presentation-space handle.

state *(BIT16)*
New state of user button.

oldstate *(BIT16)*
Old state of user button.

USHORT
Unsigned integer in the range 0 through 65 535.

VIOSIZECOUNT
Count of VIO cell sizes, see DevEscape.

maxcount *(LONG)*
Maximum number of VIO cell sizes supported.

count *(LONG)*
Number of VIO cell sizes returned.

VIOFONTCELLSIZE
VIO cell size, see DevEscape.

cx *(LONG)*
Cell width.

cy *(LONG)*
Cell height.

VOID
A data area of undefined format.

WIDTH4
Width in the range −2 147 483 648 through 2 147 483 647.

WNDPARAMS
Window parameters.

status *(BIT16)*
Window parameter selection.

Identifies the window parameters that are to be set or queried:

WPM_CBCTLDATA	Window control data length
WPM_CCHTEXT	Window text length
WPM_CTLDATA	Window control data
WPM_PRESPARAMS	Presentation parameters
WPM_TEXT	Window text.

length *(COUNT2CH)*
Length of window text.

text *(STRL)*
Window text.

presparamslength *(COUNT2B)*
Length of presentation parameters.

presparams *(PRESDATA)*
Presentation parameters.

ctldatalength *(COUNT2B)*
Length of window class specific data.

ctldata *(CTLDATA)*
Window class specific data.

WNDPROC
Window-procedure identifier.

WPOINT
Window-point structure (integer).

x *(SHORT)*
x coordinate.

dummy1 *(SHORT)*
Reserved.

y *(SHORT)*
y coordinate.

dummy2 *(SHORT)*
Reserved.

WRECT Window-rectangle structure (integer).

xleft *(SHORT)*
x coordinate of left-hand edge of rectangle.

dummy1 *(SHORT)*
Reserved.

ybottom *(SHORT)*
y coordinate of bottom edge of rectangle.

dummy2 *(SHORT)*
Reserved.

xright *(SHORT)*
x coordinate of right-hand edge of rectangle.

dummy3 *(SHORT)*
Reserved.

ytop *(SHORT)*
y coordinate of top edge of rectangle.

dummy4 *(SHORT)*
Reserved.

XYWINSIZE Window position and size structure.

x *(SHORT)*
x coordinate of window origin.

y *(SHORT)*
y coordinate of window origin.

width *(SHORT)*
Window width.

height *(SHORT)*
Window height.

window *(BIT16)*
Window options.

The values may be ORed together. For example, an invisible iconic window can be created. Note that if both XYF_MINIMIZED and XYF_MAXIMIZED are specified, the window is created in a maximized state.

XYF_INVISIBLE	Create the window initially invisible.
XYF_MAXIMIZED	Show the window initially maximized.
XYF_MINIMIZED	Show the window initially iconic.
XYF_NOAUTOCLOSE	Do not close the window automatically when the VIO application terminates. This parameter is ignored unless the program is a VIO-windowed application.
XYF_NORMAL	Create the window visible, with a size and position as specified. This is the default.

Chapter 3. Device Function Calls

The following table shows all the Device (Dev) function calls in alphabetic order according to their C/MASM name; the COBOL/FORTRAN name is also given.

C/MASM name	COBOL/FORTRAN name
DevCloseDC	WDCLS
DevEscape	WDESC
DevOpenDC	WDOPEN
DevPostDeviceModes	WDPDM
DevQueryCaps	WDQCAP
DevQueryDeviceNames	WDQDN
DevQueryHardcopyCaps	WDQHC

DevCloseDC — Close Device Context

SAA

This call closes a device context.

DevCloseDC (hdc, *hmf*)

Parameters

hdc *(HDC)* — input
Device-context handle.

hmf *(HMF)* — return
Error indicator/metafile handle (for a metafile device context).

DEV_ERROR	Error occurred
DEV_OK	Device closed, but not a metafile device context
Other	Device closed, a metafile device context whose metafile handle is returned.

Possible returns from WinGetLastError:

PMERR_NOT_CREATED_BY_DEVOPENDC
PMERR_DC_IS_ASSOCIATED
PMERR_INV_HDC

Remarks

If the device context is currently associated with a presentation space, or if it is created with the WinOpenWindowDC call (that is, it is a screen device context), an error is raised, and the device context is not closed.

If the device context being closed is a memory device context that has a bit map currently selected into it (see the GpiSetBitmap call), the bit map is automatically deselected before the device context is closed.

Any clip region currently in use for this device context is deleted.

DevEscape – Escape

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

DevEscape (hdc, Code, InCount, InData, OutCount, OutData, *Result***)**

Parameters

hdc *(HDC)* – input
Device-context handle.

Code *(LONG)* – input
Escape code.

If the device context is of any type other than OD_METAFILE or a PM_Q_STD data type spool file, then all escapes are passed through to the presentation driver. If the device context is a PM_Q_STD data type spool file, then some escapes are passed through to the presentation driver, and others are recorded in the file (this is dependent upon the escape code). For a metafile, the metafile is the presentation driver.

The description for each standard escape specifies which of these categories the escape falls into.

Devices can define additional escape functions using user **Code** values, that have the following ranges:

32 768 through 40 959 Not metafiled and not recorded (passed to presentation driver) for PM_Q_STD
40 960 through 49 151 Metafiled only (passed to presentation driver for PM_Q_STD)
49 152 through 57 343 Metafiled and recorded (not passed to presentation driver) for PM_Q_STD
57 344 through 65 535 Recorded only (not passed to presentation driver for PM_Q_STD).

The following escape functions are defined:

DEVESC_QUERYESCSUPPORT
DEVESC_GETSCALINGFACTOR
DEVESC_STARTDOC
DEVESC_ENDDOC
DEVESC_NEXTBAND
DEVESC_ABORTDOC
DEVESC_NEWFRAME
DEVESC_DRAFTMODE
DEVESC_FLUSHOUTPUT
DEVESC_RAWDATA
DEVESC_QUERYVIOCELLSIZES
DEVESC_BREAK_EXTRA
DEVESC_CHAR_EXTRA

InCount *(LONG)* – input
Input data count.

Number of bytes of data in the **InData** buffer.

InData *(BUFFER)* – input
The input data required for this escape.

DevEscape – Escape

OutCount *(LONG)* – input/output
Output data count.

OutCount is the number of bytes of data in the **OutData** buffer.

If data is actually returned in **OutData**, **OutCount** is updated to the number of bytes of data actually returned.

OutData *(BUFFER)* – output
Output data.

OutData is a buffer which receives the output from this escape. If **OutCount** is null, no data is returned.

Result *(LONG)* – return
Implemented/error indicator:

The following values can be returned:

DEVESC_ERROR	Error
DEVESC_NOTIMPLEMENTED	Escape not implemented for specified code
DEV_OK	OK.

Possible returns from WinGetLastError:

PMERR_INV_ESCAPE_CODE
PMERR_INV_HDC
PMERR_INV_LENGTH_OR_COUNT
PMERR_ESC_CODE_NOT_SUPPORTED
PMERR_INV_ESCAPE_DATA

Remarks

The data fields for standard escapes are as follows:

DEVESC_QUERYESCSUPPORT

Queries whether a particular escape call is implemented by the presentation device driver. The return value gives the result.

This escape is not metafiled or recorded.

InCount	Number of bytes pointed to by **InData**.
InData	The buffer contains an escape code value specifying the escape function to be checked.
OutCount	Not used; can be set to zero.
OutData	Not used; can be set to null.

DEVESC_GETSCALINGFACTOR

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

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

This escape is not metafiled or recorded.

InCount	Not used; can be set to zero.
InData	Not used; can be set to null.
OutCount	The number of bytes of data pointed to by **OutData**. On return, this is updated to the number of bytes actually returned.
OutData	The address of a *SFACTORS* structure, which on return contains the scaling factors for the x and y axes.

DEVESC_STARTDOC

An application must issue this escape to indicate that a new print job is starting and that all subsequent output to the device context should be spooled under the same job, until a DEVESC_ENDDOC call occurs.

If the GPI is being used in printing, this escape must be issued while the GPI presentation space is associated with the device context. This escape is metafiled but not recorded.

A GpiAssociate call must be issued to associate the presentation space with the device context before issuing this call.

InCount Number of bytes pointed to by **InData**.
InData The buffer contains a null-terminated string, specifying the name of the document.
OutCount Not used; can be set to zero.
OutData Not used; can be set to null.

DEVESC_ENDDOC

Ends a print job started by DEVESC_STARTDOC.

This escape is metafiled but not recorded.

InCount Not used; can be set to zero.
InData Not used; can be set to null.
OutCount Set equal to 2.
OutData The buffer contains a USHORT specifying the job ID if a spooler print job is created.

DEVESC_NEXTBAND

Allows an application to signal that it has finished writing to a band. The coordinates of the next band are returned. This is used by applications that handle banding (for printing) themselves.

This escape is metafiled but not recorded.

InCount Not used; can be set to zero.
InData Not used; can be set to null.
OutCount The number of bytes of data pointed to by **OutData**. On completion, this is updated to the number of bytes actually returned.
OutData The address of a *BANDRECT* structure, which on return contains the device coordinates of the next rectangular band. An empty rectangle marks the end of the banding operation.

DEVESC_ABORTDOC

Aborts the current job, erasing everything the application has written to the device since the last DEVESC_STARTDOC, including the DEVESC_STARTDOC.

This escape is metafiled but not recorded.

InCount Not used; can be set to zero.
InData Not used; can be set to null.
OutCount Not used; can be set to zero.
OutData Not used; can be set to null.

DEVESC_NEWFRAME

Allows an application to signal when it has finished writing to a page. It is similar to GpiErase processing for a screen device context, and causes a reset of the attributes. This escape is usually used with a printer device to advance to a new page.

This escape is metafiled and recorded.

InCount Not used; can be set to zero.
InData Not used; can be set to null.
OutCount Not used; can be set to zero.
OutData Not used; can be set to null.

DEVESC_DRAFTMODE

Turns draft mode on or off. Turning it on instructs the presentation device driver to print faster and with lower quality, if necessary. The draft mode can only be changed at page boundaries, for example, after a DEVESC_NEWFRAME.

To obtain maximum throughput with some printers when just printing characters, the printers must be run in text mode, not in graphics mode. The DEVESC_DRAFTMODE escape sets the printer into text mode, if it is supported. In text mode, all non-character primitives can be ignored, so any lines, arcs, areas etc, may not appear in the output. Some printers may not have the ability to position text on pel boundaries, so positioning of the text may not be perfect either. A built in font of the printer is used, and this is normally code page 437.

Only a limited number of GPI functions must actually work, the rest can be ignored. The primitive functions that will work (though possibly only with approximate positioning) are:

- GpiCharString
- GpiCharStringAt
- GpiCharStringPos
- GpiCharStringPosAt
- GpiSetCurrentPosition
- GpiMove.

All attributes can be set and queried, but many attributes, particularly color, mix, character attributes and transform rotations may not have any effect.

Note that this behavior is defined to be a "minimum." Presentation device drivers may do more if they sensibly can. Indeed they may choose to do the same as if the DEVESC_DRAFTMODE escape had not been specified (this should be the case if there are no performance advantages to limiting what they do). In this case they may even not support the DEVESC_DRAFTMODE escape.

The use of DEVESC_DRAFTMODE does not preclude the use of various hardware fonts, where these are available. The driver should take note of the **facename** on GpiCreateLogFont, noting requests for bold, italic, and point size, and substitute the most appropriate font.

This escape is metafiled and recorded.

InCount	Number of bytes pointed to by **InData**.
InData	The buffer contains a SHORT integer value specifying the mode: **1** for draft mode on **0** for draft mode off.
OutCount	Not used; can be set to zero.
OutData	Not used; can be set to null.

DEVESC_FLUSHOUTPUT

Flushes any output in the device buffer.

This escape is metafiled and recorded.

InCount	Not used; can be set to zero.
InData	Not used; can be set to null.
OutCount	Not used; can be set to zero.
OutData	Not used; can be set to null.

DEVESC_RAWDATA

Allows an application to send data directly to a presentation device driver. For example, in the case of a printer device driver, this could be a printer data stream.

If DEVESC_RAWDATA is intermingled with other data being sent to the same page of a DC, such as GPI data, the results are unpredictable, and depend upon the action taken by the Presentation Manager presentation driver. For example, a Presentation Manager device driver might ignore GPI data if DEVESC_RAWDATA is intermingled with it on the same page. In general, DEVESC_RAWDATA should be sent either to a separate page, using the DEVESC_NEWFRAME escape to obtain a new page, or to a separate document, using DEVESC_STARTDOC and DEVESC_ENDDOC escapes to create a new document.

This escape is metafiled and recorded.

InCount Number of bytes pointed to by **InData**.
InData Pointer to the raw data.
OutCount Not used; can be set to zero.
OutData Not used; can be set to null.

DEVESC_QUERYVIOCELLSIZES

Returns the VIO cell sizes supported by the presentation device driver.

This escape must be supported by presentation device drivers that provide more than two VIO cell sizes.

This escape is not metafiled or recorded.

InCount Not used; can be set to zero.
InData Not used; can be set to null.
OutCount The number of bytes of data pointed to by **OutData**. It must be an even multiple of the size in bytes of the *LONG* data type. On return, this is updated to the number of bytes actually returned.
OutData The address of a buffer, which on return contains a *VIOSIZECOUNT* structure, immediately followed by **count** copies of a *VIOFONTCELLSIZE* structure.

- If **OutCount** is less than the size of a *LONG* data type, **OutCount** is updated to zero, and nothing is returned in the buffer pointed to by **OutData**.
- If **OutCount** is equal to the size of a *LONG* data type, this is a query as to the number of VIO cell sizes that can be returned by this escape. The buffer pointed to by **OutData** is updated so that **maxcount** is the number of VIO cell sizes that can be returned.
- If **OutCount** is greater than the size of a *LONG* data type, this is a query as to the actual VIO cell sizes that are supported. The buffer pointed to by **OutData** is updated so that
  - **maxcount** is the number of VIO cell sizes that can be returned.
  - **count** is the number of VIO cell sizes actually returned (may be zero if **OutCount** is equal to twice the size of a *LONG* data type).

– **count** copies of a *VIOFONTCELLSIZE* structure are returned.

DEVESC_CHAR_EXTRA

Specifies the extra spacing value, in world coordinates. This is the amount by which the effective width of each character in a string is to be increased.

The extra width added to the break character is additional to any width vector provided with GpiCharStringPos, GpiCharStringPosAt, GpiQueryCharStringPos or GpiQueryCharStringPosAt, and to any kerning adjustments.

The extra spacing value is initialized to zero whenever a device context is opened. Any change made to the extra spacing remains in effect until either the device context is closed or a new change to the extra spacing is made.

Text drawn in a path will not be affected by extra spacing. This means that outlined text, and text used to define a clipping boundary, are constructed as if the extra spacing were set to zero.

Changing the world coordinate system after setting the extra spacing will change the amount of extra spacing in device coordinates, because the extra spacing is defined in world coordinate space.

This escape is only supported on drivers that support PostScript devices. It is metafiled and recorded.

InCount Number of bytes pointed to by **InData**.

If zero is specified, the extra spacing value is set to zero. Otherwise, this parameter must be set to exactly the size of a *FIXED* data type.

InData If **InCount** is the size of a *FIXED* data type, the **InData** buffer contains a *FIXED* number that is the new extra spacing value. This may be positive, zero, or negative.

OutCount Not used; can be set to zero.

OutData Not used; can be set to null.

DEVESC_BREAK_EXTRA

Specifies the break-extra spacing value, in world coordinates. This is the amount by which the effective width of each break character in a string is to be increased.

The code point of the break character is defined by the font. An application may determine a font's break character by issuing a GpiQueryFonts, and inspecting the metrics data returned.

The extra width added to the break character is the sum of the break-extra and character-extra amounts (see DEVESC_CHAR_EXTRA). It is also additional to any width vector provided with GpiCharStringPos, GpiCharStringPosAt, GpiQueryCharStringPos or GpiQueryCharStringPosAt, and to any kerning adjustments.

The break-extra spacing value is initialized to zero whenever a device context is opened. Any change made to the break-extra spacing remains in effect until either the device context is closed or a new change to the break-extra spacing is made.

Text drawn in a path will not be affected by break-extra spacing. This means that outlined text, and text used to define a clipping boundary, are constructed as if the break-extra spacing were set to zero.

Changing the world coordinate system after setting the break-extra spacing will change the amount of break-extra spacing in device coordinates, because the break-extra spacing is defined in world coordinate space.

This escape is only supported on drivers that support PostScript devices. It is metafiled and recorded.

InCount Number of bytes pointed to by **InData**.

If zero is specified, the break-extra spacing value is set to zero. Otherwise, this parameter must be set to exactly the size of a *FIXED* data type.

InData If **InCount** is the size of a *FIXED* data type, the **InData** buffer contains a *FIXED* number that is the new break-extra spacing value. This may be positive, zero, or negative.

OutCount Not used; can be set to zero.

OutData Not used; can be set to null.

Graphic Elements and Orders

DevEscape calls only generate orders when metafiling.

Order: Extended Escape

DevOpenDC – Open Device Context

SAA

This call creates a device context.

DevOpenDC (hab, Type, Token, Count, Data, Comp, *hdc*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

Type *(LONG)* – input
Type of device context:

OD_QUEUED	A device, such as a printer or plotter, for which output is to be queued.
OD_DIRECT	A device, such as a printer or plotter, for which output is not to be queued.
OD_INFO	A device, such as a printer or plotter, but the device context is only 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 is updated.
OD_METAFILE	The device context is used to write a metafile. The presentation page defines the area of interest within the picture in the metafile. See OD_METAFILE_NOQUERY.
OD_METAFILE_NOQUERY	The device context is used to write a metafile. Functionally, this device type is the same as OD_METAFILE, except that attribute querying is not allowed with a presentation space while it is associated with an OD_METAFILE_NOQUERY device context. If attribute querying is not required, OD_METAFILE_NOQUERY should be used in preference to OD_METAFILE, since it gives improved performance.
OD_MEMORY	A device context that is used to contain a bit map. **Comp** identifies a device with which the memory device context is to be compatible.

Token *(STRL)* – input
Device-information token.

This identifies the device information, held in the initialization file. This information is the same as that which may be pointed to by **Data**; any information that is obtained from **Data** overrides the information obtained by using this parameter.

If **Token** is specified as "*", no device information is taken from the initialization file.

OS/2 Version 1.2 behaves as if "*" is specified, but it allows any string to be specified.

Count *(LONG)* – input
Number of items.

This is the number of items present in the **Data** parameter. This may be less than the full list if omitted items are irrelevant, or supplied from **Token** or elsewhere.

Data *(DEVOPENDATA)* – input
Open-device-context data area.

Comp *(HDC)* — input
Compatible-device-context handle.

When **Type** is OD_MEMORY, this parameter is a handle to a device context compatible with bit maps that are to be used with this device context.

If **Comp** is NULL, compatibility with the screen is assumed.

hdc *(HDC)* — return
Device-context handle:

DEV_ERROR	Error
≠0	Device-context handle.

Possible returns from WinGetLastError:

PMERR_INV_DC_TYPE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_DC_DATA
PMERR_INV_HDC
PMERR_INV_DRIVER_NAME
PMERR_INV_LOGICAL_ADDRESS

Remarks

A device context is a means of writing to a particular device. Before using GPI functions to cause output to be directed to the device context, the GpiAssociate call must be issued (or the GPIA_ASSOC option specified on GpiCreatePS).

DevOpenDC cannot be used to open a device context for a screen window; use WinOpenWindowDC instead.

The device context is owned by the process from which this call is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

This call requires the existence of a message queue.

DevPostDeviceModes — Post Device Modes

This call returns, and optionally sets, printer and job properties.

DevPostDeviceModes (hab, DriverData, DriverName, DeviceName, Name, Options, *DriverCount***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

DriverData *(DRIVDATA)* — input/output
Driver data.

A data area that, on return, contains device data defined by the driver. If the pointer to the area is NULL, this call returns the required size of the data area.

The format of the data is the same as that which occurs within the *DEVOPENSTRUC* structure, passed on the **Data** parameter of DevOpenDC.

DriverName *(STRL)* — input
A string containing the name of the presentation device driver, for example, "PLOTTERS".

DeviceName *(STR32)* — input
Device-type name.

Identifies the particular device type, for example, "IBM6180" (model number). Valid names are defined by device drivers.

Note that this parameter always overrides the data in the **devicename** field of the *DRIVDATA* structure, passed on the **DriverData** parameter.

Name *(STRL)* — input
Device name.

A name that identifies the device, for example, "PRINTER1". The name is defined by the user on the Control Panel. If DPDM_POSTJOBPROP is specified on the **Options** parameter, the **Name** parameter can be NULL. For more details about specifying NULL on the **Name** parameter, see the following section.

Options *(BIT32)* — input
Dialog options.

Options that control what kind of dialog occurs, and whether the initialization file is updated.

DPDM_POSTJOBPROP Display dialog for job properties and return the job properties (do not update the initialization file).

If DPDM_POSTJOBPROP is specified, and the device driver supports job property information, **Name** can be specified as NULL. In this case the device driver takes the data passed in the **DriverData** parameter, displays it in the job properties dialog box, allows the user to change it, and passes it back to the caller. If the device driver cannot interpret the data passed in **DriverData** (for example, if it is for the wrong version), the driver defaults are used.

If the device driver does not support job property information, a NULL value for the **Name** parameter gives an error. The application must handle this case if it uses this combination of parameters.

DPDM_CHANGEPROP	Display dialogs for both job and printer properties and return the printer properties (update the initialization file).
DPDM_QUERYJOBPROP	Do not display a dialog. Return the default job properties from the initialization file.
Other	Reserved.

DriverCount *(LONG)* — return
Size/error indicator.

Value depends on what was passed as the pointer to **DriverData**:

NULL	**DriverCount** can have the following values:	
	DPDM_ERROR	Error
	DPDM_NONE	No settable options
	>0	Size in bytes required for **DriverData**.
Other	**DriverCount** can have the following values:	
	DPDM_ERROR	Error
	DPDM_NONE	No device modes
	DEV_OK	OK.

Possible returns from WinGetLastError:

PMERR_INV_DRIVER_DATA
PMERR_DRIVER_NOT_FOUND
PMERR_INV_DEVICE_NAME
PMERR_INV_LOGICAL_ADDRESS

Remarks

This call allows the user to set properties for the print job and for the device. Print quality (draft or near-letter quality), paper size for the job, and font size are examples of job properties. Current paper size, current plotter pens, and installed font cartridges are printer properties.

The application can call this 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 entered. The returned data can then be passed on DevOpenDC, as **DriverData** within the **Data** parameter.

This call requires the existence of a message queue.

DevQueryCaps — Query Device Capabilities

SAA

This call queries the device characteristics.

DevQueryCaps (hdc, Start, Count, Array, *Success***)**

Parameters

hdc *(HDC)* — input
Device-context handle.

Start *(LONG)* — input
First item of information.

The number of the first item of information to be returned in **Array**, counting from zero.

Note that in some languages, if the element constant value is used for this value, 1 must be subtracted from it. Refer to the appropriate bindings reference.

Count *(LONG)* — input
Count of items of information.

This is the count to be returned in **Array**. It must be greater than zero.

Array *(LONG****Count)** — output
Device capabilities.

Array of **Count** elements, starting with **Start**. The array elements are numbered consecutively, starting with CAPS_FAMILY. The element number constants start either with 0 or 1, depending on the language. Refer to the appropriate bindings reference.

If **Start**+**Count**—1 exceeds the current highest-defined element number, elements beyond the highest are returned as zero.

CAPS_FAMILY — Device type (values as for **Type** on DevOpenDC).

CAPS_IO_CAPS — Device input/output capability:

CAPS_IO_DUMMY
Dummy device
CAPS_SUPPORTS_OP
Device supports output
CAPS_SUPPORTS_IP
Device supports input
CAPS_SUPPORTS_IO
Device supports output and input.

CAPS_TECHNOLOGY — Technology:

CAPS_TECH_UNKNOWN
Unknown
CAPS_TECH_VECTOR_PLOTTER
Vector plotter
CAPS_TECH_RASTER_DISPLAY
Raster display
CAPS_TECH_RASTER_PRINTER
Raster printer
CAPS_TECH_RASTER_CAMERA
Raster camera
CAPS_TECH_POSTSCRIPT
PostScript device.

CAPS_DRIVER_VERSION — Presentation device driver version.

CAPS_HEIGHT	Media depth (for a full screen maximized window for displays) in pels. (For a plotter, a pel is defined as the smallest possible displacement of the pen and can be smaller than a pen width.)
CAPS_WIDTH	Media width (for a full screen maximized window for displays) in pels.
CAPS_HEIGHT_IN_CHARS	Media depth (for a full-screen maximized window for displays) in default character rows.
CAPS_WIDTH_IN_CHARS	Media width (for a full-screen maximized window for displays) in default character columns.
CAPS_HORIZONTAL_RESOLUTION	Horizontal resolution of device in pels per meter.
CAPS_VERTICAL_RESOLUTION	Vertical resolution of device in pels per meter.
CAPS_CHAR_WIDTH	Default character-box width in pels for VIO.
CAPS_CHAR_HEIGHT	Default character-box height in pels for VIO.
CAPS_SMALL_CHAR_WIDTH	Default small character box width in pels for VIO. This is zero if there is only one character-box size.
CAPS_SMALL_CHAR_HEIGHT	Default small character box height in pels for VIO. This is zero if there is only one character-box size.
CAPS_COLORS	Number of distinct colors supported at the same time, including reset (gray scales count as distinct colors). If loadable color tables are supported, this is the number of entries in the device color table. For plotters, the value returned is the number of pens plus one (for the background).
CAPS_COLOR_PLANES	Number of color planes.
CAPS_COLOR_BITCOUNT	Number of adjacent color bits for each pel (within one plane).
CAPS_COLOR_TABLE_SUPPORT	Loadable color table support: **CAPS_COLTABL_RGB_8** 1 if RGB color table can be loaded, with a minimum support of 8 bits each for red, green, and blue. **CAPS_COLTABL_RGB_8_PLUS** 1 if color table with other than 8 bits for each primary can be loaded. **CAPS_COLTABL_TRUE_MIX** 1 if true mixing occurs when the logical color table has been realized, providing that the size of the logical color table is not greater than the number of distinct colors supported (see element CAPS_COLORS). **CAPS_COLTABL_REALIZE** 1 if a loaded color table can be realized.
CAPS_MOUSE_BUTTONS	The number of pointing device buttons that are available. A returned value of 0 indicates that there are no pointing device buttons available.
CAPS_FOREGROUND_MIX_SUPPORT	Foreground mix support: **CAPS_FM_OR** Logical-OR. **CAPS_FM_OVERPAINT** Overpaint. **CAPS_FM_XOR** Logical-XOR.

CAPS_FM_LEAVEALONE
Leave alone.
CAPS_FM_AND
Logical-AND.
CAPS_FM_GENERAL_BOOLEAN
All other mix modes, see GpiSetMix.

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 CAPS_FM_OR + CAPS_FM_OVERPAINT + CAPS_FM_LEAVEALONE, 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 six bits long, with each bit set to 1 if the appropriate mode is supported.

Those mixes returned as supported are guaranteed for all primitive types. For more information, see GpiSetMix.

CAPS_BACKGROUND_MIX_SUPPORT Background mix support:

CAPS_BM_OR
Logical-OR.
CAPS_BM_OVERPAINT
Overpaint.
CAPS_BM_XOR
Logical-XOR.
CAPS_BM_LEAVEALONE
Leave alone.

The value returned is the sum of the values appropriate to the mixes supported. A device must, as a minimum, return CAPS_BM_OVERPAINT + CAPS_BM_LEAVEALONE, 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 four bits long, with each bit set to 1 if the appropriate mode is supported.

Those mixes returned as supported are guaranteed for all primitive types. For more information, see GpiSetBackMix.

CAPS_VIO_LOADABLE_FONTS Number of fonts that can be loaded for VIO.

CAPS_WINDOW_BYTE_ALIGNMENT Whether the client area of VIO windows should be byte-aligned:

CAPS_BYTE_ALIGN_REQUIRED
Must be byte-aligned.
CAPS_BYTE_ALIGN_RECOMMENDED
More efficient if byte-aligned, but not required.
CAPS_BYTE_ALIGN_NOT_REQUIRED
Does not matter whether byte-aligned.

CAPS_BITMAP_FORMATS Number of bit-map formats supported by device.

CAPS_RASTER_CAPS	Device raster operations capability: **CAPS_RASTER_BITBLT** 1 if GpiBitBlt and GpiWCBitBlt supported. **CAPS_RASTER_BANDING** 1 if this device supports banding. **CAPS_RASTER_BITBLT_SCALING** 1 if GpiBitBlt and GpiWCBitBlt with scaling supported. **CAPS_RASTER_SET_PEL** 1 if GpiSetPel supported. **CAPS_RASTER_FONTS** 1 if this device can draw raster fonts.
CAPS_MARKER_HEIGHT	Default marker box height in pels.
CAPS_MARKER_WIDTH	Default marker box width in pels.
CAPS_DEVICE_FONTS	Number of device-specific fonts.
CAPS_GRAPHICS_SUBSET	Graphics drawing subset supported.
CAPS_GRAPHICS_VERSION	Graphics architecture version number supported.
CAPS_GRAPHICS_VECTOR_SUBSET	Graphics vector drawing subset supported.
CAPS_DEVICE_WINDOWING	Device windowing support: **CAPS_DEV_WINDOWING_SUPPORT** 1 if device supports windowing. Other bits are reserved zero.
CAPS_ADDITIONAL_GRAPHICS	Additional graphics support: **CAPS_GRAPHICS_KERNING_SUPPORT** 1 if device supports kerning. **CAPS_FONT_OUTLINE_DEFAULT** 1 if device has a default outline font. **CAPS_FONT_IMAGE_DEFAULT** 1 if device has a default image font. **CAPS_SCALED_DEFAULT_MARKERS** 1 if default markers are to be scaled by the marker-box attribute. Other bits are reserved zero.
CAPS_PHYS_COLORS	Maximum number of distinct colors specifiable on device.
CAPS_COLOR_INDEX	Maximum logical color-table index supported for this device. For the EGA and VGA drivers, the value is 63.
CAPS_GRAPHICS_CHAR_WIDTH	Default graphics character box width, in pels.
CAPS_GRAPHICS_CHAR_HEIGHT	Default graphics character box height, in pels.
CAPS_HORIZONTAL_FONT_RES	Effective horizontal device resolution in pels per inch, for the purpose of selecting fonts. For printers, this is the actual device resolution, but for displays it may differ from the actual resolution for reasons of legibility.
CAPS_VERTICAL_FONT_RES	Effective vertical device resolution in pels per inch, for the purpose of selecting fonts.

DevQueryCaps — Query Device Capabilities

SAA

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HDC
PMERR_INV_QUERY_ELEMENT_NO
PMERR_INV_LENGTH_OR_COUNT

Remarks

GpiQueryDevice can be used to find the handle of the currently associated device context.

DevQueryDeviceNames — Query Device Names

This call causes a presentation driver to return the names and descriptions of the devices it supports, and their data types.

DevQueryDeviceNames (hab, DriverName, dn, DeviceName, DeviceDesc, dt, DataType, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

DriverName *(STRL)* — input
Fully-qualified name of the file containing the presentation driver.

The file name extension is '.DRV:'.

dn *(LONG)* — input/output
Maximum number of device names and descriptions that can be returned.

dn can have the following values:

Zero The number of device names and descriptions supported is returned; **DeviceName** and **DeviceDesc** are not updated.
Nonzero **dn** is updated to the number actually returned in **DeviceName** and **DeviceDesc**; **DeviceName** and **DeviceDesc** are updated.

DeviceName *(STR32****dn)** — output
Device-name array.

An array of null-terminated strings, each element of which identifies a particular device (for example, model number). Valid names are defined by presentation device drivers.

DeviceDesc *(STR64****dn)** — output
Device description array.

An array of null-terminated strings, each element of which is a description of a particular device (for example, model name). Valid descriptions are defined by presentation device drivers.

dt *(LONG)* — input/output
Maximum number of data types that can be returned.

dt can have the following values:

Zero The number of data types supported is returned, and **DataType** is not updated.
Nonzero **dt** is updated to the number actually returned, and **DataType** is updated.

DataType *(STR16****dt)** — output
Data type array.

An array of null-terminated strings, each element of which identifies a data type. Valid data types are defined by presentation device drivers.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_LENGTH_OR_COUNT

DevQueryDeviceNames – Query Device Names

Remarks

The application can call the function first with **dn** and **dt** set to zero to find how much storage is needed for the data areas. Having allocated the storage, the application then calls the function a second time for the data to be entered.

"IBM4201" is an example of a device name, "IBM 4201 Proprinter" is an example of a device description, and "PM_Q_STD" is an example of a data type.

DevQueryHardcopyCaps — Query Hardcopy Caps

This call queries the hardcopy capabilities of a device.

DevQueryHardcopyCaps (hdc, StartForm, Forms, HcInfo, *FormsReturned***)**

Parameters

hdc *(HDC)* — input
Device-context handle.

StartForm *(LONG)* — input
Start-forms code.

Forms code number from which the query is to start. The first forms code is specified by the value 0. Used with **Forms**.

Forms *(LONG)* — input
Number of forms to query.

For example, if there are five forms codes defined, **StartForm** = 2, and **Forms** = 3, a query is performed for forms codes 2, 3, and 4, and the result returned in the buffer pointed to by **HcInfo**.

If zero, the number of forms codes defined is returned. If greater than zero, this call returns the number of forms codes for which information is returned.

HcInfo *(HCINFO)* — output
Hardcopy capabilities information.

A buffer containing the results of the query. The result consists of **Forms** copies of the *HCINFO* structure.

At least one of the defined forms codes should have the HCAPS_CURRENT bit set. There may be more than one with either the HCAPS_CURRENT or the HCAPS_SELECTABLE bits set.

For a job to be selected, each one of the forms specified on the FORM spooler parameter (see **spoolerparams** in *DEVOPENSTRUC*) must be either HCAPS_CURRENT or HCAPS_SELECTABLE. The following are possibilities:

- All forms specified are HCAPS_SELECTABLE
- The single form specified is HCAPS_CURRENT
- One of the forms is HCAPS_CURRENT, and all the others are HCAPS_SELECTABLE.

FormsReturned *(LONG)* — return
Details of forms:

DQHC_ERROR	Error.
≥0	If **Forms** equals 0, number of forms available. If **Forms** does not equal 0, number of forms returned.

Possible returns from WinGetLastError:

PMERR_INV_HDC
PMERR_INV_FORMS_CODE
PMERR_INV_LENGTH_OR_COUNT

Chapter 4. Graphics Function Calls

Coordinates

GPI coordinate values that are in world or model space are passed in variables of data type *LONG*. For a presentation space of format GPIF_LONG (see GpiCreatePS), the signed value must be contained within the low-order 28 bits.

For a presentation space with a format of GPIF_SHORT, the signed value must be contained within the low-order 16 bits. Coordinates that exceed this limit are truncated without error, when stored in a segment. As a consequence, a large positive number may appear as a negative number.

In both instances, after transformation to media space (that is, device space, together with a translation in the instance of a windowed device for the window origin on the screen), coordinate values must be in the range –32 768 through +32 767.

The PMERR_COORDINATE_OVERFLOW error condition occurs if a coordinate is too large to be handled.

Region coordinates must be within the range –32 767 through +32 765.

Matrix Parameter Values

These GPI calls define transforms:

- GpiSetSegmentTransformMatrix
- GpiSetModelTransformMatrix
- GpiCallSegmentMatrix
- GpiSetViewingTransformMatrix
- GpiSetDefaultViewMatrix
- GpiCreatePS
- GpiSetPageViewport.

Programming Note: The last two calls define the device transform; the page viewport may be defaulted.

Concatenation of transform matrixes can occur as the transform is specified, for example, if TRANSFORM_ADD is specified. Concatenation also occurs during drawing, between the various transforms in the viewing pipeline.

During the process of concatenation, it is possible for the matrix parameter overflow error, PMERR_INV_MATRIX_ELEMENT, to occur. This error is raised if either of the following conditions occurs for any intermediate value during the concatenation arithmetic (see, for example, GpiSetSegmentTransformMatrix for an explanation of matrix element numbers):

- Any of the matrix elements 1, 2, 4, or 5 is greater than 32 767 or less than –32 768 (± 1 for a GPIF_SHORT format presentation space), or
- Either of elements 7 or 8 is greater than 134 217 727 ($2^{27} - 1$) or less than –134 217 728 (-2^{27}) (greater than 32 767 or less than –32 768 for a GPIF_SHORT format presentation space).

Rounding Errors

In general for graphics coordinates, when non-unity transforms (apart from simple translation) are involved, rounding errors occur. For example, adding the coordinates of one point to a delta value, to produce the coordinates of a second point (all in world coordinates) does not always map to the same device pel as if the computation had been done in device coordinates. Such errors can be avoided if calculations are done in device coordinates, or if there are no scaling (or rotational, or shear) elements in the transforms. Alternatively, the problems can be reduced, though not eliminated, by defining very fine world coordinates.

Drawing Process Check Errors

Some GPI calls involve processing buffers of graphics orders, or retained graphics segments (the data for which consists of graphics orders). These functions can all give rise to Drawing Process Check (DPC) errors, if an order is found that is not valid in its context, or which contains invalid data. If this happens, processing of the call stops, and the error is recorded. Note that orders up to the one found to be in error are processed by the function, and output occurs if drawing is being performed.

Each call that can return these errors has "DPC errors" in its error condition list. The full list of DPC errors is:

PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_IN_ELEMENT
PMERR_ALREADY_IN_ELEMENT
PMERR_STOP_DRAW_OCCURRED (warning)
PMERR_PATH_INCOMPLETE
PMERR_AREA_INCOMPLETE
PMERR_IMAGE_INCOMPLETE
PMERR_INV_ORDER_LENGTH
PMERR_NOT_IN_IMAGE
PMERR_NOT_IN_AREA
PMERR_NOT_IN_ELEMENT
PMERR_NOT_IN_PATH
PMERR_INSUFFICIENT_MEMORY
PMERR_SEG_CALL_STACK_EMPTY
PMERR_SEG_CALL_STACK_FULL
PMERR_TRUNCATED_ORDER
PMERR_CALLED_SEG_NOT_FOUND
PMERR_DYNAMIC_SEG_SEQ_ERROR
PMERR_PROLOG_ERROR
PMERR_INV_IN_VECTOR_SYMBOL

GPI Calls by Functional Area

The following table shows how all the Graphics Presentation Interface (GPI) calls are related within functional areas. The calls are in alphabetic order within these areas according to their C/MASM name; the COBOL/FORTRAN name is also given.

C/MASM name	COBOL/FORTRAN name
Arc Functions	
Attribute Setting Functions	
GpiQueryArcParams	GIQAP
GpiQueryDefArcParams	GIQDAP
GpiSetArcParams	GISAP
GpiSetDefArcParams	GIDAP
Primitive Functions	
GpiFullArc	GIFARC
GpiPartialArc	GIPARC
GpiPointArc	GIGARC
GpiPolyFillet	GIPFLT
GpiPolyFilletSharp	GIPFLS
GpiPolySpline	GIPSPL

C/MASM name	COBOL/FORTRAN name
Area Functions	
Attribute Setting Functions	
GpiQueryPattern	GSQPAT
GpiQueryPatternRefPoint	GIQPRP
GpiQueryPatternSet	GSQPTS
GpiSetPattern	GSPAT
GpiSetPatternRefPoint	GIPRP
GpiSetPatternSet	GSPATS
Primitive Functions	
GpiBeginArea	GSAREA
GpiEndArea	GSENDA
Resources and Defaults Functions	
GpiQueryBitmapHandle	GSQBTH
GpiSetBitmapId	GSBMID
Bit-Map Support	
Creation and Selection Functions	
GpiCreateBitmap	GSCBM
GpiDeleteBitmap	GSDBM
GpiLoadBitmap	GSLBM
GpiQueryBitmapDimension	GSQBTD
GpiSetBitmap	GSSBM
GpiSetBitmapDimension	GSBDIM
Operations on Raw Bit Maps	
GpiQueryBitmapBits	GSQBB
GpiQueryBitmapParameters	GSQBTP
GpiQueryDeviceBitmapFormats	GSQDBF
GpiSetBitmapBits	GSBBIT
Operations through Presentation Spaces	
GpiBitBlt	GIBB
GpiQueryPel	GIQPEL
GpiSetPel	GIPEL
GpiWCBitBlt	GIWBB
Character Functions	
Attribute Setting Functions	
GpiQueryCharAngle	GIQCA
GpiQueryCharBox	GIQCB
GpiQueryCharDirection	GSQCD
GpiQueryCharMode	GSQCM
GpiQueryCharSet	GSQCS
GpiQueryCharShear	GIQCH

GpiSetCharAngle	GICA
GpiSetCharBox	GICB
GpiSetCharDirection	GSCD
GpiSetCharMode	GSCM
GpiSetCharSet	GSCS
GpiSetCharShear	GICH
Primitive Functions	
GpiCharString	GSCHAP
GpiCharStringAt	GICHAR
GpiCharStringPos	GICHPP
GpiCharStringPosAt	GICHPR
GpiQueryCharStringPos	GIQCPO
GpiQueryCharStringPosAt	GIQCRO
Resources and Defaults Functions	
GpiCreateLogFont	GSCRLF
GpiDeleteSetId	GSRSS
GpiLoadFonts	GSFLO
GpiQueryCp	GSQCPG
GpiQueryDefCharBox	GIQCEL
GpiQueryFontFileDescriptions	GSQFD
GpiQueryFontMetrics	GSQFM
GpiQueryFonts	GSQF
GpiQueryKerningPairs	GSQKPR
GpiQueryNumberSetIds	GSQNSS
GpiQuerySetIds	GSQSS
GpiQueryTextBox	GIQTB
GpiQueryWidthTable	GSQFWT
GpiSetCp	GSCPG
GpiUnloadFonts	GSFUL
Color and Mix Functions	
Attribute Setting Functions	
GpiQueryBackColor	GSQBCO
GpiQueryBackMix	GSQBMX
GpiQueryColor	GSQCOL
GpiQueryMix	GSQMIX
GpiSetBackColor	GSBCOL
GpiSetBackMix	GSBMIX
GpiSetColor	GSCOL
GpiSetMix	GSMIX
Resources and Default Functions	
GpiCreateLogColorTable	GSCLCT

GpiQueryColorData	GSQCOD
GpiQueryColorIndex	GSQCOI
GpiQueryLogColorTable	GSQLCT
GpiQueryNearestColor	GSQNC
GpiQueryRealColors	GSQRC
GpiQueryRGBColor	GSQRGB
GpiRealizeColorTable	GSRCT
GpiUnrealizeColorTable	GSUCT
Control Functions	
GpiAssociate	GSASS
GpiCreatePS	GIPCRT
GpiDestroyPS	GIPDEL
GpiErrorSegmentData	GSESD
GpiQueryDevice	GSQD
GpiQueryPS	GIPQRY
GpiResetPS	GSRPS
GpiRestorePS	GSRSPS
GpiSavePS	GSSPS
GpiSetPS	GIPS
Correlation and Boundary Determination Functions	
Bounds Data Functions	
GpiQueryBoundaryData	GIQBD
GpiResetBoundaryData	GSRBD
Correlation Data Functions	
GpiCorrelateChain	GICORS
GpiCorrelateFrom	GICORF
GpiCorrelateSegment	GICSEG
Pick Aperture and Tag Functions	
GpiQueryDefTag	GSQDTA
GpiQueryPickAperturePosition	GIQPAP
GpiQueryPickApertureSize	GIQPAS
GpiQueryTag	GSQTAG
GpiSetDefTag	GSDTAG
GpiSetPickAperturePosition	GIPAP
GpiSetPickApertureSize	GISPAS
GpiSetTag	GSTAG
Drawing Functions	
GpiDrawChain	GSDCHN
GpiDrawDynamics	GSDDYN
GpiDrawFrom	GSDF
GpiDrawSegment	GSDSEG

GpiErase	GSERSE
GpiGetData	GSGET
GpiPutData	GSPUT
GpiQueryDrawControl	GSQDC
GpiQueryDrawingMode	GSQDM
GpiQueryStopDraw	GSQSDW
GpiRemoveDynamics	GSRDYN
GpiSetDrawControl	GSDC
GpiSetDrawingMode	GSDM
GpiSetStopDraw	GSSDW
General Attribute Functions	
Attribute Mode Functions	
GpiPop	GSPOP
GpiQueryAttrMode	GSQAM
GpiQueryDefAttrs	GIQDAT
GpiSetAttrMode	GSAM
GpiSetDefAttrs	GIDATR
Attribute Strip Setting Functions	
GpiQueryAttrs	GIQATR
GpiSetAttrs	GIATR
Image Functions	
Primitive Functions	
GpiImage	GSIMG
Line Functions	
Attribute Setting Functions	
GpiQueryLineEnd	GSQLE
GpiQueryLineJoin	GSQLJ
GpiQueryLineType	GSQLT
GpiQueryLineWidth	GIQLW
GpiQueryLineWidthGeom	GIQGLW
GpiSetLineEnd	GSLE
GpiSetLineJoin	GSLJ
GpiSetLineType	GSLT
GpiSetLineWidth	GILW
GpiSetLineWidthGeom	GIGLW
Primitive Functions	
GpiBox	GIBOX
GpiLine	GILINE
GpiMove	GIMOVE
GpiPolyLine	GIPLNE
GpiQueryCurrentPosition	GIQCP

GpiSetCurrentPosition	GICP
Visibility Functions	
GpiPtVisible	GIPVIS
GpiRectVisible	GIRVIS
Marker Functions	
Attribute Setting Functions	
GpiQueryMarker	GSQMS
GpiQueryMarkerBox	GIQMB
GpiQueryMarkerSet	GSQMSS
GpiSetMarker	GSMS
GpiSetMarkerBox	GIMB
GpiSetMarkerSet	GSMSS
Primitive Functions	
GpiMarker	GIMARK
GpiPolyMarker	GIMRKS
Metafile Support	
GpiCopyMetaFile	GSMCPY
GpiDeleteMetaFile	GSMDEL
GpiLoadMetaFile	GSMLOD
GpiPlayMetaFile	GSLOAD
GpiQueryMetaFileBits	GSMGB
GpiQueryMetaFileLength	GSQMFL
GpiSaveMetaFile	GSMSAV
GpiSetMetaFileBits	GSMFB
Miscellaneous Functions	
GpiComment	GSCOMM
Path Functions	
Path Clipping Functions	
GpiSetClipPath	GSSCLP
Path Definition and Deletion Functions	
GpiBeginPath	GSPATH
GpiCloseFigure	GSFCLS
GpiEndPath	GSENDP
Path Drawing Functions	
GpiFillPath	GSFILP
GpiOutlinePath	GSOUTP
GpiStrokePath	GSSTRP
Path Manipulation Functions	
GpiModifyPath	GSMODP

C/MASM name	COBOL/FORTRAN name
Region Support	
Clipping Region Functions	
GpiExcludeClipRectangle	GIECR
GpiIntersectClipRectangle	GIICR
GpiOffsetClipRegion	GIOCR
GpiQueryClipBox	GIQCLB
GpiQueryClipRegion	GSQCLR
GpiSetClipRegion	GSSCLR
Drawing Functions	
GpiPaintRegion	GSPREG
Region Functions	
GpiCombineRegion	GSCREG
GpiCreateRegion	GIREG
GpiDestroyRegion	GSDREG
GpiEqualRegion	GSEREG
GpiOffsetRegion	GIOREG
GpiPtInRegion	GIPIR
GpiQueryRegionBox	GIQRB
GpiQueryRegionRects	GIQRR
GpiRectInRegion	GIRIR
GpiSetRegion	GIREGN
Segment Manipulation Functions	
Segment Content Manipulation Functions	
GpiBeginElement	GSELEM
GpiDeleteElement	GSDLE
GpiDeleteElementRange	GSDLER
GpiDeleteElementsBetweenLabels	GSDELB
GpiElement	Not Used
GpiEndElement	GSENDE
GpiLabel	GSINLB
GpiOffsetElementPointer	GSOEP
GpiQueryEditMode	GSQEDM
GpiQueryElement	Not Used
GpiQueryElementPointer	GSQEP
GpiQueryElementType	GSQETS
GpiSetEditMode	GSEDM
GpiSetElementPointer	GSEP
GpiSetElementPointerAtLabel	GSEPLB
Whole Segment Functions	
GpiCloseSegment	GSSCLS

GpiDeleteSegment	GSSDEL
GpiDeleteSegments	GSDELS
GpiOpenSegment	GSSEG
GpiQueryInitialSegmentAttrs	GSQATI
GpiQuerySegmentAttrs	GSQATS
GpiQuerySegmentNames	GSQNAM
GpiQuerySegmentPriority	GSQPRI
GpiSetInitialSegmentAttrs	GSSATI
GpiSetSegmentAttrs	GSSATS
GpiSetSegmentPriority	GSSPRI
Transform Functions	
Clipping	
GpiQueryDefViewingLimits	GIQDVL
GpiQueryGraphicsField	GIQFLD
GpiQueryViewingLimits	GIQSVL
GpiSetDefViewing Limits	GISDVL
GpiSetGraphicsField	GIFLD
GpiSetViewingLimits	GISVL
Conversion Functions	
GpiConvert	GICONV
Device Transforms	
GpiQueryPageViewport	GIQPV
GpiSetPageViewport	GIPV
Helper Functions	
GpiRotate	GIROT
GpiScale	GISCAL
GpiTranslate	GIXLAT
Modelling Transform Functions	
GpiCallSegmentMatrix	GICALM
GpiQueryModelTransformMatrix	GIQCTM
GpiQuerySegmentTransformMatrix	GIQTFM
GpiSetModelTransformMatrix	GISCTM
GpiSetSegmentTransformMatrix	GISTFM
Viewing Transform Functions	
GpiQueryDefaultViewMatrix	GIQDVM
GpiQueryViewingTransformMatrix	GIQVTM
GpiSetDefaultViewMatrix	GISDVM
GpiSetViewingTransformMatrix	GISVTM

GpiAssociate – Associate

SAA

This call associates or disassociates a graphics presentation space with a device context.

GpiAssociate (hps, hdc, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

hdc *(HDC)* – input
Device-context handle.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_PS_IS_ASSOCIATED
PMERR_DC_IS_ASSOCIATED
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_HDC
PMERR_REALIZE_NOT_SUPPORTED
PMERR_PATH_INCOMPLETE (warning
PMERR_AREA_INCOMPLETE (warning

Remarks

Any type of device context may be used.

Subsequent drawing functions direct output to the associated device context.

If a null handle is supplied for the device context, the presentation space is disassociated from its currently associated device context. An associated presentation space or device context cannot be associated with another device context or presentation space, respectively.

It is an error to try to draw to a presentation space associated with a memory device context that has no bit map selected into it (see GpiSetBitmap).

The processing described for GRES_ATTRS (see GpiResetPS) is performed on the presentation space. Also, bounds data is destroyed, the page viewport is reset to its default value (see GpiCreatePS), and any clip region is lost, as is any path definition. The save/restore presentation space stack (see GpiSavePS) is purged.

Any dynamic segments left drawn on the device are not subsequently removed by GpiRemoveDynamics.

GpiBeginArea — Begin Area

This call begins the construction of an area.

GpiBeginArea (hps, Options, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(BIT32)* — input
Area options.

This contains fields of option bits. For each field, one value should be selected (unless the default is suitable). These values can then be ORed together to form the parameter, to determine whether to draw boundary lines as well as the area interior:

BA_NOBOUNDARY Do not draw boundary lines
BA_BOUNDARY Draw boundary lines (the default).
How to construct the area interior:
BA_ALTERNATE Construct interior in *alternate* mode (the default)
BA_WINDING Construct interior in *winding* mode.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_AREA_CONTROL
PMERR_INV_IN_PATH
PMERR_ALREADY_IN_AREA

Remarks

The construction is terminated by the GpiEndArea call.

The following calls can be used inside the area bracket, between this call and the following GpiEndArea call, to define the area:

- GpiBeginElement
- GpiBox (with the **Control** parameter set to DRO_OUTLINE)
- GpiCallSegmentMatrix
- GpiComment
- GpiElement (containing a valid call)
- GpiEndElement
- GpiFullArc (with the **Control** parameter set to DRO_OUTLINE)
- GpiLabel
- GpiLine
- GpiMove
- GpiPartialArc
- GpiPointArc
- GpiPolyFillet
- GpiPolyFilletSharp

- GpiPolyLine
- GpiPolySpline
- GpiPop (that pops a valid call)
- GpiSetArcParams
- GpiSetAttrs (setting only valid line attributes, or foreground color/mix (only) for other primitive types)
- GpiSetAttrMode
- GpiSetColor
- GpiSetCurrentPosition
- GpiSetLineEnd
- GpiSetLineJoin
- GpiSetLineType
- GpiSetLineWidth
- GpiSetMix
- GpiSetModelTransformMatrix.

GpiBox and GpiFullArc are only valid in an area bracket (that is, between GpiBeginArea and GpiEndArea) with the **Control** parameter set to DRO_OUTLINE. Other values of this parameter on these calls cause an implicit area bracket around the function.

The area shading is performed using the current pattern, as set by the GpiSetPattern call. The color and color-mixing modes that are current at the time GpiBeginArea is issued define the attributes to be applied to the pattern. The pattern reference point is also subjected to all of the transformations (including the model transformation) in force at the time of GpiBeginArea.

The area boundary consists of one or more **closed** figures, each constructed by:

- GpiBox
- GpiFullArc
- GpiPointArc
- GpiLine
- GpiPartialArc
- GpiPolyFilletSharp
- GpiPolyLine
- GpiPolySpline
- GpiPolyFillet.

The GpiSetColor and GpiSetMix functions can be used to control how the area boundary is to be colored. Calls to GpiSetLineEnd, GpiSetLineJoin, GpiSetLineType and GpiSetLineWidth can be interspersed to control line attributes as required. GpiSetAttrs can be used as an alternative way of setting these attributes. GpiSetArcParams can be used to control the shape of arcs produced by GpiFullArc, GpiPointArc, and GpiPartialArc.

The start of a new figure is indicated by:

- GpiCallSegmentMatrix
- GpiFullArc
- GpiMove
- GpiPop (or end of called segment), which pops current position or a model transform
- GpiSetModelTransformMatrix
- GpiSetCurrentPosition.

Programming Note: GpiCloseFigure must not be issued within an area.

A GpiBox or GpiFullArc call within an area definition generates a complete closed figure. These calls must not be used within another figure definition.

The starting point of each closed figure is the current position when this call is issued, or as specified by the call starting the figure. Figure construction continues until either a new figure is started, or the GpiEndArea function is met.

Each figure should be closed, that is, the start and end points should be identical. If this is not so, the figure is arbitrarily closed by a straight line connecting the start and end points.

The area interior is constructed either in *alternate* mode or in *winding* mode. In alternate mode, a point is determined to be within the interior 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.

In winding mode, the direction 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 not zero.

In either mode, all of the boundaries of the area are considered to be part of the interior.

If the **Options** parameter of this call is BA_NOBOUNDARY, the actual boundary lines are **not** drawn, but the shading ends at the boundaries. If the **Options** parameter specifies BA_BOUNDARY, the boundary lines and any lines added to close figures are drawn. The lines are drawn using the current line attributes, which can be changed during construction, and shading occurs within the boundaries, as noted above.

Current position is not changed by this call, but it can be changed by the moves, arcs, fillets, and lines between this call and GpiEndArea, including any used to close figures.

Area definitions can not be nested. This call and the GpiEndArea call for one area must be within the same segment.

In OS/2 Version 1.2 there is a limit of 1450 on the number of straight-line vertices that describe the area.

During correlation in nonretained mode, a hit on any function within an area returns GPI_HITS on the GpiEndArea function. GPI_HITS is not returned on any of the lines, arcs, and so on, that occur within the area definition.

Graphic Elements and Orders

Element Type: OCODE_GBAR

Order: Begin Area

GpiBeginElement – Begin Element

SAA

This call specifies the start of an element within a segment.

GpiBeginElement (hps, Type, Desc, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Type *(LONG)* – input
Type to be associated with element.

Application-defined elements should have type values in the range X'81xxxxxx' through X'FFxxxxxx' so as to avoid conflict with system-generated elements.

Desc *(STRL)* – input
Description.

Variable-length character string, recorded with the type.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_ALREADY_IN_ELEMENT
PMERR_DESC_STRING_TRUNCATED

Remarks

This call specifies the start of an element, which is stored in the current segment (in **retain** or **draw-and-retain** mode; see GpiSetDrawingMode). The element is drawn in **draw** or **draw-and-retain** mode.

The drawing calls, that are the contents of the element, are passed on subsequent GPI calls (only those calls that can generate orders are logically part of the element). The element extends up to the next GpiEndElement call (or GpiCloseSegment, which causes an implicit GpiEndElement to be generated).

Grouping drawing calls together into an element is useful if the set of calls is to be changed or replaced together at a later time. Drawing calls that are not explicitly grouped together with a GpiBeginElement/GpiEndElement pair, generate a single element for each Gpi call.

The GpiElement call, that itself generates a complete element, is not allowed within an element bracket. The GpiLabel call is also not allowed within an element bracket. Elements must not be nested within one segment.

GpiBeginElement — Begin Element

Graphic Elements and Orders

This call identifies the start of an element. The element type is defined by the **Type** parameter.

Order: Begin Element

GpiBeginPath — Begin Path

SAA

This call defines the start of a path.

GpiBeginPath (hps, Path, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Path *(LONG)* — input
Path identifier.

This must be **1**.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PATH_ID
PMERR_ALREADY_IN_PATH
PMERR_INV_IN_AREA

Remarks

Paths can be used for these purposes:

- To generate lines and curves that have a geometric width (that is, a width that is subject to transformations); see GpiModifyPath and GpiStrokePath.
- To generate lines and curves that have cosmetic width; see GpiOutlinePath. In particular, if the lines and curves are defined by characters drawn with an outline font, hollow characters are produced. (Hollow characters can also be drawn outside paths, using the FATTR_SEL_OUTLINE *FATTRS* option with GpiCreateLogFont.)
- To generate nonrectangular shapes to be used for clipping; see GpiSetClipPath.
- To generate shapes to be filled; see GpiFillPath.

  Programming Note: Areas can also be used for filling; see GpiBeginArea.

A path is specified by a number of figures, within a GpiBeginPath/GpiEndPath bracket. Each figure is specified by line calls, or curve calls, or both of these, and is separated from other figures by one of these calls:

- GpiCallSegmentMatrix
- GpiCharString
- GpiCharStringPos
- GpiCharStringAt
- GpiCharStringPosAt
- GpiFullArc
- GpiMarker
- GpiMove
- GpiPolyMarker

- GpiPop (which restores current position)
- GpiSetCurrentPosition
- GpiSetModelTransformMatrix.

A figure that is terminated by one of the calls in this list is said to be an *open* figure. A figure can also be terminated by a GpiCloseFigure call, in which instance it is said to be a *closed* figure.

A GpiBox or GpiFullArc call within a path specifies a complete closed figure. These calls must not be used within another figure specification.

GpiBeginPath initializes the path to be empty.

Path specification calls are terminated by GpiEndPath. If there are no primitives between GpiBeginPath and GpiEndPath, a null path is specified. The GpiEndPath that terminates this path specification must occur within the same segment as the GpiBeginPath.

Path specification calls can occur within a segment bracket. If the drawing mode (see GpiSetDrawingMode) is set to **draw** or **draw-and-retain**, the path is defined as it is specified. If drawing mode is **retain**, path definition does not occur until the segment containing the path specification is drawn.

The process of path definition causes a description of the path to be built in the currently associated device context. This description is used during any subsequent operation on the path. If the definition occurred by the drawing of a retained segment containing specification calls, these may subsequently be edited, with no effect on the path definition, until the segment is drawn again.

When a path has been defined, the definition cannot be reopened. An attempt to redefine the path results in the definition being replaced.

As the path definition is kept in the device context, association of the presentation space with a new device context means that the definition is lost.

When it has been defined, a path can only be used in a single GpiFillPath, GpiStrokePath, GpiOutlinePath, or GpiSetClipPath call, or modified once only with a GpiModifyPath, and then used in a single GpiFillPath, or GpiSetClipPath call. If, in a "normal" (not a micro) presentation space, a path is required to be reused, it can be created in a retained segment (for example, using **draw-and-retain** mode, see GpiSetDrawingMode), and this segment must be "drawn" whenever the definition has to be recreated. This may be done even if the application is otherwise nonretained. Otherwise, the application must reissue all of the individual calls to reconstruct the path whenever the definition has to be recreated.

A path definition is bound in device coordinates at the time the path is defined. If any transforms (other than the final windowing transform) are changed subsequently, they have no effect on the path itself (although they affect the thickness if the path is to be stroked using GpiModifyPath, or the pattern reference point if it is to be filled with GpiFillPath, or both if GpiStrokePath is used).

Line type and line width have no effect on a path. Geometric line width takes effect if the path is stroked with GpiModifyPath or GpiStrokePath.

GpiBeginPath — Begin Path

SAA

These calls can be used inside the path bracket, between this call and the following GpiEndPath call, to define the path:

- GpiBeginElement (containing only valid call(s))
- GpiBox (must specify DRO_OUTLINE option)
- GpiCallSegmentMatrix
- GpiCharString
- GpiCharStringAt
- GpiCharStringPos
- GpiCharStringPosAt
- GpiCloseFigure
- GpiComment
- GpiElement (containing a valid call)
- GpiEndElement
- GpiFullArc (must specify DRO_OUTLINE option)
- GpiLabel
- GpiLine
- GpiMarker
- GpiMove
- GpiPartialArc
- GpiPointArc
- GpiPolyFillet
- GpiPolyFilletSharp
- GpiPolyMarker
- GpiPolyLine
- GpiPolySpline
- GpiPop (if only a valid call is popped)
- GpiSetArcParams
- GpiSetAttrMode
- GpiSetAttrs
- GpiSetCharAngle
- GpiSetCharBox
- GpiSetCharDirection
- GpiSetCharMode
- GpiSetCharSet
- GpiSetCharShear
- GpiSetColor
- GpiSetCurrentPosition
- GpiSetLineEnd
- GpiSetLineJoin
- GpiSetLineType
- GpiSetLineWidth
- GpiSetMarker
- GpiSetMarkerBox
- GpiSetMarkerSet
- GpiSetMix
- GpiSetModelTransformMatrix.

The GpiCharString... calls are allowed only if the current font is an outline font. GpiQueryCharStringPos, GpiQueryCharStringPosAt, and GpiQueryTextBox are also allowed, providing the current font is an outline font.

In OS/2 Version 1.2 there is a limit of 1450 on the number of straight line vertices that describe the path. Curves are decomposed into straight lines internally, and the number of resulting vertices are also subject to this limit. The same applies to outline font character strings; if solid-filled outline characters are to be drawn, it is better to do this outside a path definition.

It is invalid for this call to occur within an area definition.

Graphic Elements and Orders

Element Type: OCODE_GBPTH

Order: Begin Path

GpiBitBlt — Bit Blt

This call copies a rectangle of bit-map image data.

GpiBitBlt (Target, Source, Count, Points, Rop, Options, *Hits*)

Parameters

Target *(HPS)* — input
Target presentation-space handle.

Source *(HPS)* — input
Source presentation-space handle.

Count *(LONG)* — input
Point count.

Number of points specified in **Points**.

If this is 3, a source rectangle of the same size as the target rectangle is used. If it is 4, stretching or compression is performed as necessary. If compression is performed, the **Options** parameter determines how eliminated rows or columns are handled.

Points *(POINT*Count)* — input
Point array.

Array of **Count** points, in the order **Tx1, Ty1, Tx2, Ty2, Sx1, Sy1, Sx2, Sy2**. These are:

Tx1,Ty1 Specify the bottom-left corner of the target rectangle in target device coordinates.
Tx2,Ty2 Specify the top right-corner of the target rectangle in target device coordinates.
Sx1,Sy1 Specify the bottom-left corner of the source rectangle in source device coordinates.
Sx2,Sy2 Specify the top-right corner of the source rectangle in source device coordinates (not required if neither stretching nor compression is to be performed).

Rop *(LONG)* — input
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 the 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, and T pel values, the final target value for the pel is determined by the appropriate **Rop** bit value as shown in the table below:

P	S	T (initial)	T (final)
0	0	0	Index bit 0 (least significant)
0	0	1	Index bit 1
0	1	0	Index bit 2
0	1	1	Index bit 3
1	0	0	Index bit 4
1	0	1	Index bit 5
1	1	0	Index bit 6
1	1	1	Index bit 7 (most significant)

The index formed in the above way determines the mixing required. Mnemonic names are available for commonly-used mixes:

```
ROP_SRCCOPY       /* SRC                     */
ROP_SRCPAINT      /* SRC OR DST              */
ROP_SRCAND        /* SRC AND DST             */
ROP_SRCINVERT     /* SRC XOR DST             */
ROP_SRCERASE      /* SRC AND NOT(DST)        */
ROP_NOTSRCCOPY    /* NOT(SRC)                */
ROP_NOTSRCERASE   /* NOT(SRC) AND NOT(DST)   */
ROP_MERGECOPY     /* SRC AND PAT             */
ROP_MERGEPAINT    /* NOT(SRC) OR DST         */
ROP_PATCOPY       /* PAT                     */
ROP_PATPAINT      /* NOT(SRC) OR PAT OR DST  */
ROP_PATINVERT     /* DST XOR PAT             */
ROP_DSTINVERT     /* NOT(DST)                */
ROP_ZERO          /* 0                       */
ROP_ONE           /* 1                       */
```

Options *(BIT32)* — input
Options.

How eliminated lines or columns are treated if a compression is performed.

Bits 15 through 31 of **Options** may be used for privately-supported modes for particular devices.

BBO_OR	The default. If compression is necessary, logical-OR eliminated rows or columns. This is useful for white on black.
BBO_AND	If compression is necessary, logical-AND eliminated rows or columns. This is useful for black on white.
BBO_IGNORE	If compression is necessary, ignore eliminated rows or columns. This is useful for color.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_BITBLT_MIX
PMERR_INV_BITBLT_STYLE
PMERR_BITMAP_NOT_FOUND
PMERR_INV_COORDINATE
PMERR_INV_RECT
PMERR_NO_BITMAP_SELECTED
PMERR_INCORRECT_DC_TYPE
PMERR_INCOMPATIBLE_BITMAP

Remarks

A rectangle of bit-map image data is copied from a bit map selected into a device context associated with the source presentation space, to a bit map selected into a device context associated with the target presentation space. Alternatively, either presentation space may be associated with a device context that specifies a suitable raster device, for example, the screen.

Programming Note: In either case, both source and target device contexts must apply to the same physical device. It is an error if this 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 is nondestructive when source and target rectangles overlap.

A rectangle can be specified in device coordinates, for both source and target. These rectangles are noninclusive; that is, 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 pel as the top-right, that rectangle is deemed to be empty.

If the top-right source point is specified, and the source and target rectangles are of different sizes, stretching, or compressing, or both, of the data occurs. **Options** specifies how eliminated rows or columns of bits are to be treated if compression occurs. Note that the pattern data is never stretched or compressed.

These current attributes of the target presentation space are used (other than for converting between monochrome and color, as described below):

Area color
Area background color
Pattern set
Pattern symbol.

The color values are used in conversion between monochrome and color data. This is the only format conversion performed by this call. The conversions are:

- Output of a monochrome pattern to a color device.

  In this instance, the pattern is converted first to a color pattern using the current area colors:

  - source 1s → area foreground color
  - source 0s → area background color.

- Copying from a monochrome bit map to a color bit map (or device).

  The source bits are converted as follows:

  - source 1s → image foreground color
  - source 0s → image background color.

- Copying from a color bit map to a monochrome bit map (or device).

  - source pels that are the source image background color → image background color.
  - all other pels → image foreground color.

Programming Note: In all of the above instances (except where the source image background color is used) it is the attributes of the *target* presentation space that are used.

If the mix (**Rop**) does not call for a pattern, the pattern set and pattern symbol are not used. If it does not require a source (this is invalid when **Options** is in the range 1 through 3), **Source** is not required and must be null. **Sx1,Sy1** is also ignored in this instance.

Neither the source nor the pattern is required when a bit map, or part of a bit map, is to be cleared to a particular color.

If the mix does require both source and pattern, a three-way operation is performed.

If a pattern is required, dithering may be performed for solid patterns in a color that is not available on the device. See GpiSetPattern.

If any of the source data is not available (when, for example, the source presentation space is connected to a screen window, and the source rectangle is not totally visible), the contents of the unavailable parts are undefined. This can be checked with GpiRectVisible before calling this function.

This call is independent of drawing mode (see GpiSetDrawingMode); the effect always occurs immediately, and it is not retained even if the drawing mode is **draw-and-retain** or **retain**. (Its effect, however, is recorded in a metafile, but note that this is only successful if the metafile is replayed on a similar device, with **draw** drawing mode.)

The current position in both source and target presentation spaces is unchanged by this call.

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiBox — Box

This call draws a rectangular box with diagonally opposite corners at the current position and the specified position.

GpiBox (hps, Control, Point, HRound, VRound, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Control *(LONG)* — input
Outline and fill control.

Specifies whether the interior of the box is to be filled, and if the outline is to be drawn:

DRO_FILL	Fill interior
DRO_OUTLINE	Draw outline
DRO_OUTLINEFILL	Draw outline and fill interior.

Point *(POINT)* — input
Corner point.

The coordinates of the diagonally opposite corner.

HRound *(ROL)* — input
Corner-rounding control.

Horizontal length of the **full** axis of the ellipse used for rounding at each corner.

VRound *(ROL)* — input
Corner-rounding control.

Vertical length of the **full** axis of the ellipse used for rounding at each corner.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_BOX_CONTROL
PMERR_INV_COORDINATE
PMERR_INV_BOX_ROUNDING_PARM

Remarks

The sides of the box are parallel to the x and y axes, in world coordinates.

The four corners of the box can be rounded with a quarter ellipse. The size of this ellipse is specified by **HRound** and **VRound**. If **HRound** equals **VRound**, the corners of the box are rounded with a quarter circle.

If either **HRound** or **VRound** is zero, no rounding occurs.

If current position is (x0,y0), and **Point** is set to (x1,y1), the box is drawn from (x0,y0) to (x1,y0) to (x1,y1) to (x0,y1) to (x0,y0). This is significant in such cases as area winding mode; see GpiBeginArea.

Current position is unchanged by this call.

Either the outline of the box, or its interior, or both, can be drawn.

If this call occurs within an area or path definition, it generates a complete closed figure (DRO_OUTLINE must be specified). It must not occur within any other figure definition.

If correlation is in force, a hit always results if the pick aperture intersects the box boundary. However, if the pick aperture lies wholly within the box, a hit only occurs if the interior is being drawn (DRO_FILL or DRO_OUTLINEFILL).

Graphic Elements and Orders

Element Type: OCODE_GCBOX

Order: Box at Current Position

GpiCallSegmentMatrix — Call Segment Matrix

This function calls a segment, and specifies the instance transform with which it is called.

GpiCallSegmentMatrix (hps, Segment, Count, Array, Options, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Segment *(LONG)* — input
Identifier of segment to be called.

This must be greater than 0.

The specified segment must not be a chained segment.

Count *(LONG)* — input
Number of elements.

The number of elements of **Array** to be examined, starting from the beginning of the structure. If **Count** is less than 9, remaining elements default to the corresponding elements of the identity matrix. If **Count** = 0, the identity matrix is used.

Array *(MATRIX)* — input
Instance transform matrix.

The third, sixth, and ninth elements, when specified, must be 0, 0, and 1, respectively.

Options *(LONG)* — input
Transformation options.

Specifies how the transform defined by the **Array** transform should be used to modify the existing current model transform for the duration of the call (the existing transform is the concatenation, in the current call context, of the instance, segment and model transforms, from the root segment downwards):

TRANSFORM_REPLACE	Previous model transform is discarded and replaced by the specified transform.
TRANSFORM_ADD	Specified transform is combined with the existing model transform, in the order (1) existing transform, (2) new transform. This option is most useful for incremental updates to transforms.
TRANSFORM_PREEMPT	Specified transform is combined with the existing model transform, in the order (1) new transform, (2) existing transform.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_TRANSFORM_TYPE
PMERR_CALLED_SEG_NOT_FOUND

PMERR_CALLED_SEG_IS_CHAINED
PMERR_CALLED_SEG_IS_CURRENT
PMERR_SEG_CALL_STACK_EMPTY

Remarks

The instance transform specified is a model transform that is used to modify the current model transform, in a way that depends upon the value of the **Options** parameter, before calling the segment. This new transform only applies to the called segment. On return, it is reset to the model transform in operation before the call was made.

The transform is specified as a one-dimensional array of elements, being the first **Count** elements of a 3-row by 3-column matrix ordered by rows. The order of the elements is:

```
Matrix                  Array

┌       ┐
│ a  b  0 │
│ c  d  0 │          (a,b,0,c,d,0,e,f,1)
│ e  f  1 │
└       ┘
```

A point with coordinates (x,y) is transformed to the point

(a*x + c*y + e, b*x + d*y + f)

The called segment must have a unity transform for the viewing transform (see GpiSetViewingTransformMatrix).

If scaling values greater than unity are given (which only applies if the presentation space coordinate format as set by the GpiCreatePS call is GPIF_LONG), it is possible for the combined effect of this and any other relevant transforms to exceed fixed-point implementation limits. This causes an error.

Graphic Elements and Orders

Element Type: OCODE_GCALLS

Order: Push and Set Model Transform
Order: Call Segment
Order: Pop

GpiCharString — Character String

This call draws a character string starting at the current position.

GpiCharString (hps, Count, String, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of characters.

The maximum number is 512.

String *(STR)* — input
Characters to be drawn.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_FONT_AND_MODE_MISMATCH

Remarks

Each character in the string is positioned so that its character reference point is at the current position. The current position is advanced after each character is drawn to give the position for the next character.

The characters in the character string are selected from the current character-set. The particular font from which the characters are selected depends on the current character mode. For a description of which fonts are used for each of the possible modes, see GpiSetCharMode.

The degree to which approximation of the position and size of characters is allowed, and also the area used during correlation of the character string, is controlled by the character-mode attribute.

After the string has been drawn, the current position is set to the end of the character string. This is the point at which the next character would have been drawn, had it existed.

Graphic Elements and Orders

Element Type: OCODE_GCCHSTM

Order: Character String Move at Current Position

GpiCharStringAt — Character String At

SAA

This call draws a character string starting at the specified position.

GpiCharStringAt (hps, Point, Count, String, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — input
Starting position.

Defines the position at which the first character in the string is to be placed, in world coordinates.

Count *(LONG)* — input
Number of characters.

The maximum number is 512.

String *(STR)* — input
Characters to be drawn.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_INV_LENGTH_OR_COUNT
PMERR_FONT_AND_MODE_MISMATCH

Remarks

The call GpiCharStringAt (hps, start, count, string) is equivalent to:

```
GpiMove (hps, start)
GpiCharString (hps, count, string)
```

Each character in the string is positioned so that its character reference point is at the current position. The current position is advanced after each character is drawn to give the position for the next character.

The font from which the characters in the character string are selected depends on the current character mode. For a description of which fonts are used for each of the possible modes, see GpiSetCharMode.

The degree to which approximation of the position and size is allowed, and also the area used during correlation of the character string, is controlled by the character-mode attribute.

After the string has been drawn, the current position is set to the end of the character string. This is the point at which the next character would have been drawn, had it existed.

Graphic Elements and Orders

Element_Type: OCODE_GCHSTM

Order: Character String Move at Given Position

GpiCharStringPos — Character String Position

SAA

This call draws a character string starting at the current position, with various controls including control over individual character positioning.

GpiCharStringPos (hps, Rect, Options, Count, String, Adx, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Rect *(RECT)* — input
Rectangle structure.

Defines the two corners of the rectangle, which defines the background of the characters, in world coordinates. It is ignored if neither CHS_OPAQUE nor CHS_CLIP is specified.

Options *(BIT32)* — input
Formatting options.

Option flags that can be used in combination:

CHS_OPAQUE	Background of characters is defined by the rectangle specified by **Rect**. The rectangle is to be shaded (with background color and overpaint) before drawing.
CHS_VECTOR	Increments vector is supplied (**Adx**). If zero, **Adx** is ignored.
CHS_LEAVEPOS	Leave current position at the start of the string. If not set, current position is moved to the position at which the next character would have been drawn, had there been one.
CHS_CLIP	Clips the string to the rectangle, if set.
	Other bits are reserved and must be zero.

Count *(LONG)* — input
Number of characters in the string.

The maximum number is 512.

String *(STR)* — input
Character string.

Adx *(ROL*Count)* — input
Increment values.

Vector of increment values, in world coordinates.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CHAR_POS_OPTIONS
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_RECT
PMERR_FONT_AND_MODE_MISMATCH

Remarks

A vector of increments can be specified, allowing control over the positioning of each character after the first. This vector consists of distances measured in world coordinates (along the baseline for left-to-right and right-to-left character directions, and along the shearline for top-to-bottom and bottom-to-top). The i'th increment is the distance of the reference point 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, when specified, set the widths of each character.

A further option allows a rectangle to be specified, which is used as the background of the string instead of the normal background. This rectangle is painted, using the current character background color and an overpaint mix (unless this is in a dynamic segment, when leave-alone is used). Both corners of the rectangle are specified, so that the rectangle is positioned independently of current position. Points on the borders of the rectangle are considered to be included within the rectangle.

Clipping of the string to the rectangle is also allowed. This is independent of whether the rectangle is actually drawn.

Current position can be updated to the point at which the next character would have been drawn, had there been one, or it can be left at the start of the string.

Graphic Elements and Orders

Element Type: ETYPE_GCCHSTE

Order: Character String Extended at Current Position

GpiCharStringPosAt — Character String Position At

SAA

This call draws a character string, starting at the specified position, with various controls including control over individual character positioning.

GpiCharStringPosAt (hps, Start, Rect, Options, Count, String, Adx, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Start *(POINT)* — input
Starting position.

Rect *(RECT)* — input
Rectangle structure.

Defines the two corners of the rectangle, which defines the background of the characters, in world coordinates. It is ignored unless CHS_OPAQUE or CHS_CLIP is selected.

Options *(BIT32)* — input
Formatting options:

Option flags that can be used in combination:

CHS_OPAQUE	Background of characters is defined by the rectangle specified by **Rect**. The rectangle is to be shaded (with background color and overpaint) before drawing.
CHS_VECTOR	Increments vector is supplied (**Adx**). If zero, **Adx** is ignored.
CHS_LEAVEPOS	Leave current position at the start of the string. If not set, current position is moved to the position at which the next character would have been drawn, had there been one.
CHS_CLIP	Clips the string to the rectangle, if set.
	Other bits are reserved and must be zero.

Count *(LONG)* — input
Number of characters in the string.

The maximum number is 512.

String *(STR)* — input
Character string.

Adx *(ROL****Count)*** — input
Increment values.

Vector of increment values, in world coordinates.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CHAR_POS_OPTIONS
PMERR_INV_COORDINATE
PMERR_INV_RECT
PMERR_INV_LENGTH_OR_COUNT
PMERR_FONT_AND_MODE_MISMATCH

Remarks

A vector of increments can be specified, allowing control over the position of each character after the first. This vector consists of distances measured in world coordinates (along the baseline for left-to-right and right-to-left character directions, and along the shearline for top-to-bottom and bottom-to-top). The i'th increment is the distance of the reference point (for example, 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.

A further option allows a rectangle to be specified that can be used as the background of the string instead of the normal background. This rectangle is painted using the current character background color and an overpaint mix (unless this is in a dynamic segment, when leave-alone is used). Both corners of the rectangle are specified, so that the rectangle is positioned independently of current position. Points on the borders of the rectangle are considered to be included within the rectangle.

Clipping of the string to the rectangle is also allowed. This is independent of whether the rectangle is actually drawn.

Current position can be updated to the point at which the next character would have been drawn, had there been one, or it can be left at the start of the string.

Graphic Elements and Orders

Element Type: ETYPE_GCHSTE

Order: Character String Extended at Given Position

GpiCloseFigure — Close Figure

SAA

This call closes a figure within a path specification.

GpiCloseFigure (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

The current figure is closed by a line drawn to the start point of the figure.

This call need not be used if the path is to be filled (see GpiFillPath), or used as a clip path (see GpiSetClipPath), as any figures in the path that have not been closed are automatically closed at that time. It should be used, however, for any closed figures within paths that are subsequently to be stroked by GpiModifyPath or GpiStrokePath.

This call must not be used outside a path specification (in particular, it must not be used within an area).

Graphic Elements and Orders

Element Type: OCODE_GCFIG

Order: Close Figure

GpiCloseSegment — Close Segment

This call closes the current segment.

GpiCloseSegment (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NOT_IN_SEG
PMERR_PATH_INCOMPLETE (warning
PMERR_AREA_INCOMPLETE (warning

Remarks

Closing a segment does not delete the segment or affect the graphics primitives that are drawn.

Any attributes that have been preserved (see the AM_PRESERVE option of GpiSetAttrMode), are popped (restored), when the GpiCloseSegment call is issued in **draw** or **draw-and-retain** modes, and at the end of the segment when it is subsequently drawn in **draw-and-retain** or **retain** modes (see GpiSetDrawingMode).

If an area or path is open when a segment is closed, the area or path is terminated. When the drawing mode is **draw** or **draw-and-retain**, a warning is given, but the close processing continues. No warning is given for **retain**. If a retained segment with one of these unended brackets is subsequently drawn, an error is raised.

If an element bracket is open when a segment is closed, the element bracket is first closed automatically.

If this call is followed by primitives or attributes, without first opening a segment, the following may or may not have been reset to their default values:

- Current attribute values and arc parameters
- Current tag
- Current model transform
- Current position
- The current clip path and viewing limits.

Any such quantity can 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. An application should not be written, so as to depend on the values of these quantities immediately after GpiCloseSegment.

Subsequent primitives, not preceded by an GpiOpenSegment call, are not retained, irrespective of the current drawing mode.

The current viewing transform, however, is guaranteed to be reset to unity for primitives outside segments.

GpiCombineRegion — Combine Region

This call combines two regions.

GpiCombineRegion (hps, Dest, Src1, Src2, Mode, *Complexity*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The regions must be owned by the device identified by the currently associated device context.

Dest *(HRGN)* — input
Handle of destination region.

Src1 *(HRGN)* — input
Handle of first source region.

Src2 *(HRGN)* — input
Handle of second source region.

Mode *(LONG)* — input
Method of combination:

CRGN_OR	Union of **Src1** and **Src2**
CRGN_COPY	**Src1** only (**Src2** ignored)
CRGN_XOR	Symmetric difference of **Src1** and **Src2**
CRGN_AND	Intersection of **Src1** and **Src2**
CRGN_DIFF	**Src1** AND NOT(**Src2**).

Complexity *(LONG)* — return
Complexity of resulting region/error indicator:

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_REGION_MIX_MODE
PMERR_HRGN_BUSY

Remarks

Source and destination regions must all be of the same device class. The destination region can be one of the source regions.

It is invalid if any of the specified regions are currently selected as the clip region (by GpiSetClipRegion).

GpiComment — Comment

SAA

This call adds a comment to the current segment.

GpiComment (hps, Length, Data, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Length *(LONG)* — input
Data length.

The length of **Data** in bytes. **Length** must not be greater than 255.

Data *(BUFFER)* — input
Comment data.

No conversion of any kind is performed on the data.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT

Remarks

An application can use this call to store some data of its own in the segment, if the drawing mode (see GpiSetDrawingMode) is set to **retain** or **draw-and-retain**. It has no effect on drawing. The data can be retrieved subsequently by the application using GpiGetData or GpiQueryElement.

Graphic Elements and Orders

Element Type: OCODE_GCOMT

Order: Comment

GpiConvert — Convert

This call converts an array of (x,y) coordinate pairs from one coordinate space to another.

GpiConvert (hps, Src, Targ, Count, Points, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Src *(LONG)* — input
Source-coordinate space.

Targ *(LONG)* — input
Target-coordinate space.

Count *(LONG)* — input
Point count.

Number of coordinate pairs in **Points**.

Points *(POINT*__Count__*)* — input/output
Array of (x,y) coordinate pair structures.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORD_SPACE
PMERR_COORDINATE_OVERFLOW

Remarks

The array contains x1, y1, x2, y2,.... The input coordinates are replaced by the converted coordinates.

Valid values for **Src** and **Targ** are:

CVTC_WORLD	World coordinates
CVTC_MODEL	Model space
CVTC_DEFAULTPAGE	Page space before default viewing transform
CVTC_PAGE	Page space after default viewing transform
CVTC_DEVICE	Device space.

Conversions involving either world coordinates or model space should not be performed if the drawing mode (see GpiSetDrawingMode) is **retain**.

GpiCopyMetaFile — Copy Metafile

SAA

This call creates a new metafile, and copies the contents of an existing loaded metafile into it.

GpiCopyMetaFile (hmf, *New*)

Parameters

hmf *(HMF)* — input
Source metafile handle.

New *(HMF)* — return
New metafile handle/error indicator:

≠0	New metafile handle
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HMF
PMERR_METAFILE_IN_USE
PMERR_TOO_MANY_METAFILES_IN_USE

Remarks

The source metafile must already be loaded or generated, and identified by a metafile handle. The new metafile is also identified by a handle, which is returned (it may be used, for example, by GpiPlayMetaFile).

The new metafile is owned by the process from which this call is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

GpiCorrelateChain — Correlate Chain

This call performs a correlate operation on the retained segment chain, and returns data for each tagged primitive that intersects the current aperture, as set by GpiSetPickApertureSize.

GpiCorrelateChain (hps, Type, Pick, MaxHits, MaxDepth, SegTag, *NumHits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Type *(LONG)* — input
Segment type.

Specifies the type of segment on which correlation is to be performed:

PICKSEL_VISIBLE	Only visible and detectable segments with nonzero identifiers are correlated.
PICKSEL_ALL	All segments with nonzero identifiers are correlated, regardless of the detectability and visibility attributes of the segments.

Pick *(POINT)* — input
Pick position.

The position of the aperture center in presentation page units.

MaxHits *(LONG)* — input
Maximum hits.

Maximum number of hits that may be returned in the **SegTag** parameter.

MaxDepth *(LONG)* — input
Number of pairs.

Number of segment/tag pairs to be returned by each hit.

SegTag *(LONG*2*MaxDepth*MaxHits)* — output
Segment identifiers and tags.

An array consisting of segment identifiers and primitive tags in alternate elements. For each hit, a set of **MaxDepth** segment identifiers and tag pairs is returned.

NumHits *(LONG)* — return
Number of hits, or error:

≥0	Number of hits that occurred
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_INV_MAX_HITS
PMERR_INV_CORRELATE_DEPTH
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_CORRELATE_TYPE
DPC errors

Remarks

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 that called that segment, repeated until either the root segment is reached, or **MaxDepth** segment/tag pairs are returned.

Only primitives with a nonzero tag in segments with a nonzero identifier are correlated using this call. Primitives in segments called (to any depth in the hierarchy) from an unnamed segment are not eligible 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 **MaxDepth** values, the remaining values are set to zero. If more than **MaxDepth** values are available, only that number is returned.

The number of hits that occurred is returned in **NumHits**.

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 **SegTag** parameter returned. The **NumHits** parameter, therefore, returns this distinct number of hits. Hits are returned in the reverse order of their occurrence.

SegTag is set to the hits that are found, up to the maximum defined in the **MaxHits** parameter. Corresponding pairs of elements form the "hit" pairs. The number returned by the call therefore contains the number of sets of **MaxDepth** pairs set if the **MaxHits** parameter is greater than the number of hits detected. The number of elements set in the **SegTag** parameter is twice the number returned by the call (subject to a maximum of **MaxHits**) multiplied by the **MaxDepth**.

If the **NumHits** value returned by the call is greater than that specified in **MaxHits**, more hits occurred than could be returned. If all hits are important, specify an array that is large enough to contain the maximum number of sets of hits that are expected.

The draw controls (see GpiSetDrawControl) are ignored by this call.

It may be necessary to ensure that attributes, model transform, current position, and viewing limits are reset to their default values, before processing the chain. This can be done by either ensuring that the first segment to be correlated does not have the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), or by issuing GpiResetPS before the GpiCorrelateChain. The latter method also resets the clip path to no clipping, which may also be necessary.

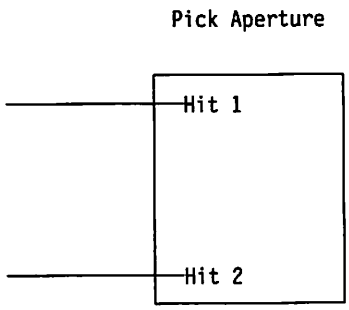
If this call is followed by primitives or attributes, without first opening a segment, the processing is as described for GpiCloseSegment.

Examples

```
Start segment 1
  Tag 10
  Call 2
End segment 1

Start segment 2
  Tag 20
  Call 3
  Tag 21
  ......
  ......
End segment 2

Start segment 3
  Tag 30
  ......
  ......
End segment 3
```



For MaxHits = 1 at MaxDepth = 2:

segment	tag
2	21
1	10

Returned **NumHits** = 2.

For MaxHits = 2 at MaxDepth = 4:

segment	tag	
2	21	hit1.1
1	10	hit1.2
0	0	hit1.3
0	0	hit1.4
3	30	hit2.1
2	20	hit2.2
1	10	hit2.3
0	0	hit2.4

Returned **NumHits** = 2.

GpiCorrelateFrom — Correlate From

SAA

This call performs a correlate operation on a section of the retained segment chain.

GpiCorrelateFrom (hps, FirstSegment, LastSegment, Type, Pick, MaxHits, MaxDepth, SegTag, *NumHits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

FirstSegment *(LONG)* — input
Specifies the first segment to be correlated.

It must be greater than 0.

LastSegment *(LONG)* — input
Specifies the last segment to be correlated.

It must be greater than 0.

Type *(LONG)* — input
The type of segments on which correlation is to be performed:

PICKSEL_VISIBLE	Only visible and detectable segments with nonzero identifiers are correlated.
PICKSEL_ALL	All segments with nonzero identifiers are correlated, regardless of the detectability and visibility attributes of the segments.

Pick *(POINT)* — input
Pick position.

The position of the aperture center in presentation page units.

MaxHits *(LONG)* — input
Maximum hits.

Maximum number of hits that can be returned in the **SegTag** parameter.

MaxDepth *(LONG)* — input
Number of pairs.

Number of segment/tag pairs to be returned by each hit.

SegTag *(LONG*2*MaxDepth*MaxHits)* — output
An array of segment and tag identifiers in alternate elements.

For each hit, a set of **MaxDepth** segment identifiers and tag pairs is returned.

NumHits *(LONG)* — return
Number of hits, or error:

≥0	Number of hits that occurred
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CORRELATE_TYPE
PMERR_INV_COORDINATE
PMERR_INV_MAX_HITS
PMERR_INV_CORRELATE_DEPTH
PMERR_INV_MICROPS_FUNCTION

PMERR_SEG_NOT_FOUND
PMERR_SEG_NOT_CHAINED
PMERR_INV_SEG_NAME
DPC errors

Remarks

The correlation operation starts at the segment identified by **FIrstSegment**, and includes chained and called segments up to, and including, the segment identified by **LastSegment**.

Data is returned for each tagged primitive that intersects the pick 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 that called the segment, repeated until the root segment is reached, or **MaxDepth** values are returned.

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 not eligible 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 **MaxDepth** values, the remaining values are set to zero. If more than **MaxDepth** values are available, only that number is returned.

The number of hits that occurred is returned in **NumHits**.

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 **SegTag** parameter returned. The number of hits return value, therefore, returns this distinct number of hits. Hits are returned in reverse order of their occurrence.

SegTag is set to the hits that are found, up to the maximum defined in the **MaxHits** parameter. Corresponding pairs of elements form the hit pairs. The number returned by the call therefore contains the number of sets of **MaxDepth** pairs set if the **MaxHits** parameter is greater than the number of hits detected. The number of elements set in the **SegTag** parameter is twice the number returned by the call (subject to a maximum of **MaxHits**) multiplied by the **MaxDepth**.

If the **NumHits** value returned by the call is greater than that specified in **MaxHits**, more hits occurred than could be returned. If all hits are important, specify an array that is large enough to contain the maximum number of sets of hits that are expected.

The draw controls (see GpiSetDrawControl) are ignored by this call.

It may be necessary to ensure that attributes, model transform, current position, and viewing limits are reset to their default values, before processing the segments. This can be done either ensuring that the first segment to be correlated does not have the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), or by issuing GpiResetPS before the GpiCorrelateFrom. The latter method also resets the clip path to no clipping, which may also be necessary.

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

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

GpiCorrelateSegment — Correlate Segment

SAA

This call performs a correlate operation on the specified segment.

```
GpiCorrelateSegment (hps, Segment, Type, Pick, MaxHits, MaxDepth, SegTag, NumHits)
```

Parameters

hps *(HPS)* — input
Presentation-space handle.

Segment *(LONG)* — input
Identifier of the segment to be correlated; it must be greater than 0.

Type *(LONG)* — input
Type of segments on which correlation is to be performed:

PICKSEL_VISIBLE	Only visible and detectable segments with nonzero identifiers are correlated.
PICKSEL_ALL	All segments with nonzero identifiers are correlated, regardless of the detectability and visibility attributes of the segments.

Pick *(POINT)* — input
Position of the center of the pick aperture in presentation page space.

MaxHits *(LONG)* — input
Maximum hits.

The maximum number of hits that may be returned in the **SegTag** parameter.

MaxDepth *(LONG)* — input
Number of segment and tag pairs to be returned for each hit.

SegTag *(LONG*MaxDepth*MaxHits)* — output
Array.

An array consisting of segment identifiers and primitive tags in alternate elements. For each hit, a set of **MaxDepth** segment identifiers and tag pairs is returned.

NumHits *(LONG)* — return
Number of hits, or error:

≥0	Number of hits that occurred
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CORRELATE_TYPE
PMERR_INV_COORDINATE
PMERR_INV_MAX_HITS
PMERR_INV_CORRELATE_DEPTH
PMERR_INV_MICROPS_FUNCTION
PMERR_SEG_NOT_FOUND
PMERR_INV_SEG_NAME
DPC errors

Remarks

Data is returned for each tagged primitive that intersects the pick aperture. The data returned for each "hit" (or correlation) consists of a set of segment and tag pairs, starting with the correlated one, the one which called that segment, repeated until the specified segment (which was not called by another segment) is reached, or **MaxDepth** values are returned.

The specified segment identifier must be nonzero. Only primitives with a nonzero tag (see GpiSetTag) are correlated using this call.

The depth value specifies the number of sets of segment and tag pairs to be returned for each hit. If the specified segment is reached before **MaxDepth** values, the remaining values are set to zero. If more than **MaxDepth** values are available, only that number is returned.

The number of hits that occurred is returned in **NumHits**.

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 **SegTag** parameter returned. The number of hits return value, therefore, returns this distinct number of hits. Hits are returned in reverse order of their occurrence.

SegTag is set to the hits that are found, up to the maximum defined in the **MaxHits** parameter. Corresponding pairs of elements form the hit pairs. The number returned by the call, therefore, contains the number of sets of **MaxDepth** pairs set if the **MaxHits** parameter is greater than the number of hits detected. The number of elements set in the **SegTag** parameter is twice the number returned by the call (subject to a maximum of **MaxHits**) multiplied by the **MaxDepth**.

If the **NumHits** value returned by the call is greater than that specified in **MaxHits**, more hits occurred than could be returned. If all hits are important, specify an array that is large enough to contain the maximum number of sets of hits that are expected.

The draw controls (see GpiSetDrawControl) are ignored by this call. This call differs from the other GpiCorrelate... calls in that the segment to be correlated need not be a chained segment.

It may be necessary to ensure that attributes, model transform, current position, and viewing limits are reset to their default values, before processing the segment. This can be done either by ensuring that the segment to be correlated does not have the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), or by issuing GpiResetPS before the GpiCorrelateSegment. The latter method also resets the clip path to no clipping, which may also be necessary.

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

GpiCreateBitmap — Create Bit Map

SAA

This call creates a bit map and returns the bit-map handle.

GpiCreateBitmap (hps, New, Options, InitData, InfoTable, *hbm*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The associated device should, if possible, hold the bit map in its own memory. Where this is not possible, main memory is used and the bit map is held in a format compatible with the device.

New *(BITMAPINFOHEADER)* — input
Bit-map information header.

This structure defines the format of the bit map to be created.

Options *(BIT32)* — input
Options:

CBM_INIT Initialize the bit map with **InitData**

If the bit map is stored on a device, the **Options** parameter is passed to the device. Bits 16 through 31 can be used for special features known to be supported by the particular device driver.

InitData *(BUFFER)* — input
Buffer address.

The address in application storage from which initialization data is to be copied, if CBM_INIT is set.

InfoTable *(BITMAPINFO)* — input
Bit-map information table.

This defines the format of the data in **InitData**. Ignored if CBM_INIT is not set.

hbm *(HBITMAP)* — return
Bit-map handle/error indicator:

≠0 New bit-map handle
GPI_ERROR Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_INFO_TABLE
PMERR_INV_USAGE_PARM

Remarks

On some devices it is possible to create the bit map in device memory. Even where this is not possible, a bit map always 'belongs' to a particular device. The device is specified through the device context associated with the specified presentation space. The device context can be any device context connected to the device in question (such as any window device context for the screen).

There are a number of standard bit-map formats that should normally be adhered to. Other formats can be used if supported by the device.

A newly created bit map can be filled with data supplied by the application. This is useful where the bit map always contains, or always starts with, the same image, captured in the application. A bit-map information structure is also passed, which defines the format and color usage of the initialization data. It is assumed that enough data is passed to initialize the entire bit map.

Some bit-map calls, including drawing into the bit map, require it to be selected into a memory device context, using GpiSetBitmap. This is true whether device or main memory is used to hold the bit map.

The bit map is owned by the process from which this call is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

GpiCreateLogColorTable – Create Logical Color Table

SAA

This call defines the entries of the logical color table.

GpiCreateLogColorTable (hps, Options, Format, Start, Count, Table, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Options *(BIT32)* – input
Options:

LCOL_RESET The color table is to be reset to default before processing the remainder of the data in this call.

This value is assumed if the color table is currently in RGB mode, and is being changed to index mode; that is, LCOLF_INDRGB or LCOLF_CONSECRGB is specified.

The **Format** parameter must be LCOLF_INDRGB or LCOLF_CONSECRGB.

LCOL_REALIZABLE The application may issue GpiRealizeColorTable at an appropriate time. This may affect the way the system maps the indexes when the logical color table is not realized.

If this option is not set, GpiRealizeColorTable will have no effect.

LCOL_PURECOLOR The application does not want color dithering to create colors not available in the physical palette for solid patterns (see GpiSetPattern). If this option is set, only pure colors are used and no dithering is done.

Other flags are reserved and must be B'0'.

Format *(LONG)* – input
Format of entries in the table:

LCOLF_INDRGB Array of (index,RGB) values. Each pair of entries is 8 bytes long, 4 bytes (local format) index, and 4 bytes color value.

This sets the color table into index mode (and forces LCOL_RESET) if it is in RGB mode.

The maximum index that may be loaded is returned in the CAPS_COLOR_INDEX parameter of the DevQueryCaps call.

LCOLF_CONSECRGB Array of (RGB) values, corresponding to color indexes **Start** upwards. Each entry is 4 bytes long.

This sets the color table into index mode (and forces LCOL_RESET) if it is in RGB mode.

The maximum index that can be loaded is returned in the CAPS_COLOR_INDEX parameter of the DevQueryCaps call.

LCOLF_RGB Color index = RGB.

This sets the color table into RGB mode.

Start *(LONG)* – input
Starting index.

This is only relevant for LCOLF_CONSECRGB.

Count *(LONG)* — input
Count of elements in **Table**.

This must be greater than or equal to zero.

If zero is specified, LCOLF_INDRGB and LCOLF_CONSECRGB have the same effect.

For LCOLF_INDRGB **Count** must be an even number.

Table *(LONG*Count)* — input
Start of the application data area.

This contains 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 is red intensity value
G is green intensity value
B is blue intensity value.

(as there are 8 bits for each primary). The maximum intensity for each primary is 255.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COLOR_OPTIONS
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_START_INDEX
PMERR_REALIZE_NOT_SUPPORTED

Remarks

This call can cause the color table to be preset to the default values. These are:

CLR_BACKGROUND	Reset color, set by GpiErase. This is the natural background color for the device. For a display, it is the default window color (SYSCLR_WINDOWTEXT; see WinSetSysColors). For a printer, it is the paper color. It can be changed by setting new system colors from the control panel for the display, or by selecting a paper color for a printer (if allowed by the device driver).
CLR_BLUE	Blue.
CLR_RED	Red.
CLR_PINK	Pink (magenta).
CLR_GREEN	Green.
CLR_CYAN	Cyan (turquoise).
CLR_YELLOW	Yellow.
CLR_NEUTRAL	A device-dependent color, that provides a contrasting color to CLR_BACKGROUND. For a display, it is the default window text color (SYSCLR_WINDOWTEXT; see WinSetSysColors). For a printer, it is a color that contrasts with the paper color. It can be changed by setting new system colors from the control panel for the display, or by selecting a paper color for a printer (if allowed by the device driver).
CLR_DARKGRAY	Dark gray.
CLR_DARKBLUE	Dark blue.

CLR_DARKRED	Dark red.
CLR_DARKPINK	Dark pink.
CLR_DARKGREEN	Dark green.
CLR_DARKCYAN	Dark cyan.
CLR_BROWN	Brown.
CLR_PALEGRAY	Pale gray.

GpiErase clears to color CLR_BACKGROUND.

By default, presentation spaces have a logical color table consisting of the 16 default values given above. In index mode, these entries are always considered as part of the color table, unless they are explicitly overwritten. Color indexes outside this range, which have not been loaded, are not considered as part of the color table; such colors must not be used if the color table is in index mode.

If LCOL_REALIZABLE is set, all drawing takes place using the index mapping appropriate to the realized table, even when the table is not realized. This includes line drawing, character drawing, and any bit maps set by GpiSetBitmapBits. The colors in the picture, when it is not realized, may therefore differ markedly from what is required.

When the table is subsequently realized, however, the colors appear as correctly as is possible on the device. If the logical color table can be completely loaded to the device, the physical color indexes used are such that color mixing is predictable (that is, a pel with a logical color index of 2 when OR-mixed with one of logical color index 1 produces a pel of logical color index 3), if the device technology allows this; this would not, for example, be true on a plotter. However, if the logical color table is too big to be loaded to the device, the system performs a mapping, and mixing is not predictable. The maximum size that may be loaded to the device can be found from DevQueryCaps (CAPS_COLORS); CAPS_COLOR_TABLE_SUPPORT (also returned by DevQueryCaps) indicates whether color mixing is predictable.

If LCOL_REALIZABLE is not set, the system performs a mapping from the colors in the logical color table to those in the standard physical color table for that device. This mapping is used for all drawing, bit maps, and so on. Mixing is not predictable.

The standard physical color table always includes the standard 16 colors, where this is physically possible. On devices that support more than 16 colors, there may be additional colors available, to which the requested colors may be mapped. However, it cannot be ensured that these additional colors are the same on different devices. Applications that depend upon precise colors beyond the first 16 should *realize* their color tables on devices for which this is supported, unless they can tell from the color queries that the colors they require are indeed available without realizing.

If LCOL_REALIZABLE is set, but the currently associated device does not support realization, a warning is raised (not an error; the value TRUE is returned). However, in this instance, the realizable flag is kept (by the system) logically set in the color table, although the color table is treated as nonrealizable while associated with a device context that does not support realization. If the presentation space is reassociated, the same warning is given if the new device does not support realization. If the device does support realization, however, the bit is honored.

For a monochrome device (whether it is a display, bit map, printer, or some other type), a reset color is defined as follows. Start with the appropriate one of:

- The paper color, for example, for a printer with no loaded color table
- SYSCLR_WINDOW for a monochrome display with no loaded color table
- Color 0 for any device if a color table has been loaded.

If this color is white, or a light color, white is chosen as the reset color; otherwise, the reset color is black. The reset color is used for:

- the color that GpiErase erases to
- CLR_BACKGROUND (color 0), unless an RGB color table is in use
- CLR_DEFAULT for GpiSetBackColor
- any color that has exactly the same RGB value as the reset color.

Any other color becomes black if the reset color is white, and the converse.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiCreateLogFont — Create Logical Font

SAA

This call provides a logical definition of a font that the application requires to use.

GpiCreateLogFont (hps, Name, Lcid, Attrs, *Match*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Name *(STR8)* — input
Logical font name.

An 8-character name that can be used to describe the logical font. Its principal use is in interchange files, where it can help to identify the required font. For example, it can reference a file name that contains the font for a remote system.

Lcid *(LONG)* — input
Local identifier.

The local identifier that the application uses to refer to this font. It must be in the range 1 through 254.

It is an error if **Lcid** is already in use to refer to a font or bit map.

Attrs *(FATTRS)* — input
Attributes of font.

The required attributes of the font.

Match *(LONG)* — return
Match indicator:

FONT_MATCH	Font requirements matched successfully
FONT_DEFAULT	Font requirements not matched; a default font is used
GPI_ERROR	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SETID
PMERR_INV_FONT_ATTRS
PMERR_FONT_NOT_LOADED
PMERR_SETID_IN_USE
PMERR_KERNING_NOT_SUPPORTED

Remarks

The system uses the available physical font that most closely matches the requirements. Physical fonts can be:

- Loaded at initialization time
- Built into particular devices or device drivers
- Private ones for this process, loaded by GpiLoadFonts.

An application can force selection of a particular physical font by quoting the **match** value in *FATTRS*, to be that returned for the desired font by GpiQueryFonts.

Whichever method is used, the choice of physical font, which is made when this call is issued, is never subsequently changed for a particular logical font.

The local identifier (**Lcid**), that the application decides to use to reference this logical font for later drawing operations, is also specified; see GpiSetCharSet.

If the face name is provided, GpiCreateLogFont tries to select the font with that face name. If the face name is empty, GpiCreateLogFont selects a default font.

When a match number is provided, GpiCreateLogFont tries to find a font with the same match number and face name. If there is a mismatch at this point, GpiCreateLogFont acts as though the match number is zero and starts the search again.

When the match number is zero and the calling program requests a bit-mapped font (FATTR_FONTUSE_OUTLINE not set), GpiCreateLogFont searches for a bit-mapped font with the required average character width (AveCharWidth), maximum baseline extent (MaxBaselineExt), consistent with the usage flags. If this search fails, GpiCreateLogFont searches for an outline font with the required face name.

When the match number is zero and the calling program requests an outline font (FATTR_FONTUSE_OUTLINE is set), GpiCreateLogFont searches for an outline font with the required selection flags. If that search fails, a default outline font is selected. If the match number is set to a positive number, a Presentation Manager font is selected. If the match number is negative, a font belonging to a physical device is selected.

It is advisable to set the values of **all** the elements in the **Attrs** structure. This is particularly important where printing, plotting, or interchange are concerned, as the target machine may need to substitute an existing device font for the requested font.

To anticipate possible substitution by a vector font, values should be set for character angle, character shear and character box (using GpiSetCharAngle, GpiSetCharShear, and GpiSetCharBox respectively) before drawing any character strings. The GpiQueryFontMetrics call can be used to get the values of the character box height and width for a font, which are held in the fields **maxbaselineext** and **avecharwidth** in the FONTMETRICS structure.

Outline font characters are normally drawn filled. However, hollow characters are produced if the FATTR_SEL_OUTLINE flag is set in the **Attrs** parameter. For small characters, outlining in this way can give a similar visual appearance to filled characters, with improved performance.

There are restrictions on the use of non installed fonts with certain device types. See GpiLoadFonts for more details.

If this call occurs within a path definition when the drawing mode (see GpiSetDrawingMode) is **retain** or **draw-and-retain**, its effect is not stored with the definition.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiCreatePS — Create Presentation Space

SAA

This call creates a presentation space.

GpiCreatePS (hab, hdc, Size, Options, *hps*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

hdc *(HDC)* — input
Device-context handle.

The handle of a device context with which the presentation space is to be associated, if GPIA_ASSOC is specified. This is mandatory for a micro-presentation space (type GPIT_MICRO).

Size *(SIZEROL)* — input
Presentation-page size.

The size of the presentation page defines a rectangle in presentation page space, with the bottom-left corner at the origin. This rectangle is used for these purposes:

- Together with the page viewport, it defines the device transform. Whenever the presentation space is associated with a device context, a default page viewport is constructed, based on the presentation page size.
- It defines the "area of interest" of the picture. This is recorded in a metafile, if one is generated from this presentation space. Note, however, that depending upon the device transform, information drawn outside it may sometimes be visible; it is *not* a clipping boundary.
- If PU_ARBITRARY is specified, the page viewport is constructed such that the origin of the page rectangle maps to the origin of the default device rectangle (maximized window size, paper size, and so on), and either the right or top edges map, keeping the picture within the default device rectangle, and preserving its aspect ratio.

If zero is specified as either the width or the height, GPIA_ASSOC must also be specified, and a presentation page of default dimension for the device (see above) is assumed. For PU_ARBITRARY the pel dimensions are used.

Options — input
Options.

This contains fields of option bits. For each field, one value should be selected (unless the default is suitable). These values can then be ORed together to generate the parameter.

PS_UNITS *(BIT6)*
Presentation-page size units

In each instance, the origin is at the bottom left.

One of these values *must* be specified:

PU_ARBITRARY	Application-convenient units
PU_PELS	Pel coordinates
PU_LOMETRIC	Units of 0.1 mm
PU_HIMETRIC	Units of 0.01 mm
PU_LOENGLISH	Units of 0.01 inch
PU_HIENGLISH	Units of 0.001 inch
PU_TWIPS	Units of 1/1440 inch.

PS_FORMAT *(BIT4)*
Coordinate format.

Indicates options to be used when storing coordinate values internally in the segment store.

For most calls, the format is not directly visible to an application. However, it is visible during editing (for example, GpiQueryElement). The format also has an effect on the amount of storage required for segment store. If a metafile is generated from this presentation space, the format also controls the format of the orders in the metafile.

Programming Note: If GPIF_SHORT is selected, it is the application's responsibility to ensure that the values passed for graphics coordinates are in the range –32 768 through +32 767, when the drawing mode (see GpiSetDrawingMode), or if a metafile is being created, is **retain** or **draw-and-retain**. If in doubt, default or specify GPIF_LONG.

Do not specify GPIF_SHORT if a metafile of unknown format is to be played into this presentation space with GpiPlayMetaFile.

One of these can be selected, for a GPIT_NORMAL presentation space (for a GPIT_MICRO presentation space, only GPIF_DEFAULT is allowed):

GPIF_DEFAULT	Default local format (same as GPIF_LONG)
GPIF_SHORT	2-byte integers
GPIF_LONG	4-byte integers.

PS_TYPE *(BIT1)*
Presentation space.

GPIT_NORMAL	Normal PS; this is the default
GPIT_MICRO	Micro-PS.

Programming Note: GPIA_ASSOC must also be set if GPIT_MICRO is set.

PS_MODE *(BIT1)*
Mode. Reserved, must be zero (default).

PS_ASSOCIATE *(BIT1)*
Association indicator.

Indicates whether the new presentation space is to be associated with the specified device context:

GPIA_NOASSOC	No association is required. This is the default.
GPIA_ASSOC	Association with **hdc** required.

Programming Note: GPIA_ASSOC must be set if GPIT_MICRO is set.

hps *(HPS)* — return
Presentation-space handle:

≠0	Presentation-space handle
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_OR_INCOMPAT_OPTIONS
PMERR_DC_IS_ASSOCIATED
PMERR_INV_HDC
PMERR_INV_PS_SIZE

GpiCreatePS — Create Presentation Space

Remarks

There are two types of presentation spaces:

- micro-presentation space
- normal presentation space.

Only a restricted subset of calls is allowed to a micro-presentation space; the main difference is that graphic segments (primitives, attributes, and so on) can be retained by the system, for subsequent redraw or editing, in a normal presentation space, but not in a micro-presentation space. However, the storage and execution overheads are lower for a micro-presentation space.

An initial association of the new presentation space with a device context may be performed (this is mandatory for a micro-presentation space), by specifying GPIA_ASSOC.

When a presentation space is associated with a device context, either using this call with GPIA_ASSOC, or explicitly with GpiAssociate, a page viewport in device space is automatically constructed, to which the page is mapped to form the device transform. The value of **PS_UNITS** and the **Size** parameter, are taken into account.

GpiCreateRegion — Create Region

This call creates a region, for a particular class of device, using a series of rectangles.

GpiCreateRegion (hps, Count, Rectangles, *hrgn*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

A region suitable for use with the currently associated device is created.

Count *(LONG)* — input
The number of rectangles.

The number specified in **Rectangles**. If **Count** is 0, an empty region is created, and **Rectangles** is ignored.

Rectangles *(RECT*Count)* — input
An array of rectangles.

The rectangles are specified in device coordinates.

For each rectangle in the array, the value of **xright** must be greater than (or equal to) **xleft**, and **ytop** must be greater than (or equal to) **ybottom**.

hrgn *(HRGN)* — return
Region handle:

≠0	Region handle
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORDINATE
PMERR_INV_RECT

Remarks

The new region is defined by the logical-OR of all of the rectangles specified. Points on the right-hand and top boundaries are not included in the region. Points on the left-hand and bottom boundaries, that are not also on the right-hand or top boundaries (that is, the top-left and bottom-right corner points), are included.

The region is owned by the process from which this call is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

GpiDeleteBitmap — Delete Bit Map

SAA

This call deletes a bit map.

```
GpiDeleteBitmap (hbm, Success)
```

Parameters

hbm *(HBITMAP)* — input
Handle of bit map to be deleted.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HBITMAP
PMERR_BITMAP_IS_SELECTED
PMERR_HBITMAP_BUSY

GpiDeleteElement — Delete Element

This call deletes the element indicated by the element pointer.

GpiDeleteElement (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NOT_IN_RETAIN_MODE
PMERR_NO_CURRENT_SEG
PMERR_INV_IN_ELEMENT

Remarks

The element pointer is set to the element immediately preceding the deleted element.

If the element pointer has a value of 0 (points that are logically before the first element), nothing is deleted and the element pointer is not changed.

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress. It is invalid within an element bracket.

GpiDeleteElementRange — Delete Element Range

SAA

This call deletes all elements between, and including, the elements indicated by the specified element numbers.

GpiDeleteElementRange (hps, FirstElement, LastElement, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

FirstElement *(LONG)* — input
Number of the first element to be deleted.

LastElement *(LONG)* — input
Number of the last element to be deleted.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NOT_IN_RETAIN_MODE
PMERR_NO_CURRENT_SEG
PMERR_INV_IN_ELEMENT

Remarks

If either element number is outside the range of the current segment, it is set to the nearest valid value.

When this call has finished, the element pointer is set to the element immediately preceding the deleted element or elements.

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress. It is not valid within an element bracket.

GpiDeleteElementsBetweenLabels — Delete Elements Between Labels

This call deletes all elements between, but not including, the elements found to contain the indicated labels.

GpiDeleteElementsBetweenLabels (hps, FirstLabel, LastLabel, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

FirstLabel *(LONG)* — input
Label marking the start of the elements to be deleted.

LastLabel *(LONG)* — input
Label marking the end of the elements to be deleted.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NOT_IN_RETAIN_MODE
PMERR_NO_CURRENT_SEG
PMERR_INV_IN_ELEMENT
PMERR_LABEL_NOT_FOUND

Remarks

The search for **FirstLabel** and **LastLabel** is performed separately, and starts from the element pointed to by the current element pointer.

See also:

- GpiSetElementPointer
- 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.

On completion, the element pointer is set to the element immediately preceding the deleted elements.

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress. It is not valid within an element bracket.

GpiDeleteMetaFile — Delete Metafile

SAA

This call deletes a metafile.

GpiDeleteMetaFile (hmf, *Success*)

Parameters

hmf *(HMF)* — input
Metafile handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HMF
PMERR_METAFILE_IN_USE
PMERR_TOO_MANY_METAFILES_IN_USE

Remarks

This call deletes access to the specified memory metafile and makes the metafile handle invalid.

GpiDeleteSegment — Delete Segment

This call deletes a retained segment.

GpiDeleteSegment (hps, Segid, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Segid *(LONG)* — input
Segment identifier.

The identifier of the segment to be deleted; it must be greater than 0.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_MICROPS_FUNCTION

Remarks

If the segment is open when it is deleted, there is no open segment after this call. In this instance, processing as described for GpiCloseSegment is performed.

If the segment is in the segment chain, it is removed from the chain.

This call deletes only a retained segment.

Programming Note: In **draw** drawing mode (see GpiSetDrawingMode), the identifier of the current segment is not remembered, so it is not recognized if specified as the **Segid** parameter.

GpiDeleteSegments — Delete Segments

SAA

This call deletes all segments in the given identifier range.

GpiDeleteSegments (hps, FirstSegment, LastSegment, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

FirstSegment *(LONG)* — input
First identifier in the range; it must be greater than 0.

LastSegment *(LONG)* — input
Last identifier in the range; it must be greater than 0.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_MICROPS_FUNCTION

Remarks

FirstSegment and **LastSegment** can have the same value, in which instance, only this segment is deleted. If **FirstSegment** is greater than **LastSegment** only the segment with identifier **FirstSegment** is deleted.

If one of the segments deleted is the currently open segment, there is no open segment after this call. In this instance, processing as described for GpiCloseSegment is performed. If any of the segments are in the segment chain, they are removed from the chain.

This call only deletes retained segments.

Programming Note: In **draw** drawing mode (see GpiSetDrawingMode), the identifier of the current segment is not remembered, so it is not recognized if it occurs within the range of specified identifiers.

GpiDeleteSetId — Delete Set Identifier

This call deletes a logical font or bit-map tag.

GpiDeleteSetId (hps, lcid, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

lcid *(LONG)* — input
Local identifier.

The local identifier (lcid) for the object.

If LCID_ALL is specified, **all** logical fonts are deleted, and all bit-map tagging is removed.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SETID
PMERR_SETID_NOT_FOUND
PMERR_SETID_IN_USE

Remarks

If the object is a logical font, it is deleted, and is no longer available for use. If the object is a bit map, it is no longer tagged with the local identifier; the bit map is not deleted and its handle remains valid.

In either instance, the **lcid** is released and is now available for reuse, unless the object is currently selected (as the current character, pattern, or marker set), in which instance an error is raised.

If this call occurs within a path definition when the drawing mode (see GpiSetDrawingMode) is **retain** or **draw-and-retain**, its effect is not stored with the definition.

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiDestroyPS — Destroy Presentation Space

SAA

This call destroys the presentation space.

GpiDestroyPS (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_PS_IS_ASSOCIATED

Remarks

All resources owned by the presentation space are released, and any subsequent calls that use the value of the presentation space handle are rejected.

GpiDestroyRegion — Destroy Region

This call destroys a region.

GpiDestroyRegion (hps, hrgn, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The region must be owned by the device identified by the currently associated device context.

hrgn *(HRGN)* — input
Handle of region to be destroyed.

If this is NULL, the call takes no action, and completes without error.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION
PMERR_HRGN_BUSY

Remarks

This call cannot be used to destroy the clip region; the clip region must first be deselected with GpiSetClipRegion.

GpiDrawChain — Draw Chain

SAA

This call draws the segments that are in the segment chain.

GpiDrawChain (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
DPC errors

Remarks

The segments drawn are all of the retained segments that have the ATTR_CHAINED segment attribute (see GpiSetInitialSegmentAttrs), together with all of the unchained segments that are called from them.

The drawing operation is controlled by the calls 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, and this call is followed by primitives or attributes, without first opening a segment, the processing is as described for GpiCloseSegment.

If a segment is already open at the time of the draw, GpiCloseSegment processing is not performed at the completion of the draw (except that any unclosed path or area is abandoned with an error). In this instance, if the open segment is the last one drawn (and no dynamic segments had to be drawn), attributes and other parameters are in the correct state to continue drawing in any drawing mode.

Dynamic segments are not drawn if they are found while processing the segment chain. However, depending on the setting of DCTL_DYNAMIC (see GpiSetDrawControl), they may be removed before, and drawn after, the operation.

It may be necessary to ensure that attributes, model transform, current position, and viewing limits are reset to their default values, before processing the chain. This can be done by ensuring that the first segment to be drawn does not have the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), or by issuing GpiResetPS before the GpiDrawChain. The latter method also resets the clip path to no clipping, which may also be necessary.

It is an error to issue this call while any of these brackets are open:

- Area bracket
- Path bracket
- Element bracket.

GpiDrawDynamics — Draw Dynamics

This call redraws the dynamic segments in, or called from, the segment chain.

GpiDrawDynamics (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_FOR_THIS_DC_TYPE
DPC errors

Remarks

Dynamic segments are those segments in the segment chain that have the ATTR_DYNAMIC segment attribute (see GpiSetInitialSegmentAttrs). It is preferable to position dynamic segments at the start of the segment chain.

Dynamic segments can either be drawn with this call, or by setting the DCTL_DYNAMIC draw control (see GpiSetDrawControl), and issuing one of the other GpiDraw... calls.

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

Notes:

1. The redraw is controlled by the calls set by previous calls to GpiSetDrawControl.
2. 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 before draw" is set ON (see GpiSetDrawControl), the presentation space is erased before the redraw.

It may be necessary to ensure that attributes, model transform, current position, and viewing limits are reset to their default values, before processing the segments. This can be accomplished either by ensuring that the first dynamic segment to be drawn does not have the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), or by issuing GpiResetPS before the GpiDrawDynamics. The latter method also resets the clip path to no clipping, which may also be necessary.

If this call is followed by primitives or attributes, without first opening a segment, then the processing is as described for GpiCloseSegment. In particular, note that during GpiDrawDynamics, the system forces the foreground mix to FM_XOR and the background mix to BM_LEAVEALONE. It may be necessary to set one or both of these before proceeding to draw.

GpiDrawFrom — Draw From

SAA

This call draws a section of the segment chain.

GpiDrawFrom (hps, FirstSegment, LastSegment, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

FirstSegment *(LONG)* — input
First segment to be drawn; it must be greater than zero.

LastSegment *(LONG)* — input
Last segment to be drawn; it must be greater than zero.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_SEG_NOT_FOUND
PMERR_SEG_NOT_CHAINED
PMERR_INV_SEG_NAME
DPC errors

Remarks

Drawing starts at the segment identified by **FirstSegment** and includes all chained segments (those with the ATTR_CHAINED segment attribute, see GpiSetInitialSegmentAttrs), and the segments called from them, up to, and including, the segment identified by **LastSegment**.

The drawing operation is controlled by the calls 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, and this call is followed by primitives or attributes, without first opening a segment, the processing is as described for GpiCloseSegment.

If a segment is already open at the time of the draw, GpiCloseSegment processing is not performed at the completion of the draw (except that any unclosed path or area is terminated with an error). In this instance, if the open segment is the last one drawn (and no dynamic segments had to be drawn), attributes and other parameters are in the correct state to continue drawing in any drawing mode.

Dynamic segments are not drawn if they are found while processing the segment chain. However, depending on the setting of DCTL_DYNAMIC (see GpiSetDrawControl), they may be removed before, and drawn after, the operation. If this happens, then **all** dynamic segments are involved, whether they occur within the range specified or not.

It may be necessary to ensure that attributes, model transform, current position, and viewing limits are reset to their default values, before processing the segments. This can be done either by ensuring that the first segment to be drawn does not have the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), or by issuing GpiResetPS before the GpiDrawFrom. The latter method also resets the clip path to no clipping, which may also be necessary.

It is an error to issue this call while any of these brackets are open:

- Area bracket
- Path bracket
- Element bracket.

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

GpiDrawSegment — Draw Segment

SAA

This call draws the specified segment.

GpiDrawSegment (hps, Segment, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Segment *(LONG)* — input
Segment to be drawn; it must be greater than zero.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_SEG_NOT_FOUND
PMERR_INV_SEG_NAME
DPC errors

Remarks

The drawing operation is controlled by the calls 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, and this call is followed by primitives or attributes, without first opening a segment, the processing is as described for GpiCloseSegment.

If a segment is already open at the time of the draw, GpiCloseSegment processing is not performed at the completion of the draw (except that any unclosed path or area is abandoned with an error). In this instance, if the open segment is the segment specified in **Segment**, and no dynamic segments had to be drawn, then attributes and other parameters are in the correct state to continue drawing in any drawing mode.

Depending on the setting of DCTL_DYNAMIC (see GpiSetDrawControl), all of the dynamic segments in the chain may be removed before, and drawn after, the specified segment is drawn. (Note that if the specified segment is itself dynamic, it is only drawn in this way.)

This call differs from the other GpiDraw... calls, in that the segment to be drawn need not be a chained segment.

It may be necessary to ensure that attributes, model transform, current position, and viewing limits are reset to their default values, before processing the segment. This can be done either by ensuring that the segment does not have the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), or by issuing GpiResetPS before the GpiDrawSegment. The latter method also resets the clip path to no clipping, which may also be necessary.

It is an error to issue this call while any of these brackets are open:

- Area bracket
- Path bracket
- Element bracket.

GpiElement — Element

This call adds a single element to the current segment.

GpiElement (hps, Type, Desc, Length, Data, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Type *(LONG)* — input
Type to be associated with the element.

Application-defined elements should have type values in the range X'81xxxxxx' through X'FFxxxxxx' so as to avoid conflict with system-generated elements.

Desc *(STRL)* — input
Element description.

This is a variable length character string that is recorded with the element.

Length *(LONG)* — input
Length of content data for the element.

This must not be greater than 63KB.

Data *(BUFFER)* — input
Buffer pointer.

Element content data.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_LENGTH_OR_COUNT
PMERR_DATA_TOO_LONG
PMERR_ALREADY_IN_ELEMENT
DPC errors

Remarks

The element is stored in the current segment if the drawing mode (see GpiSetDrawingMode) is **retain** or **draw-and-retain**. It is drawn if the drawing mode is **draw** or **draw-and-retain**.

It is an error if the element data contains any begin or end element orders. Similarly, this call is not valid within an element bracket.

Programming Note: No coordinate conversion is performed by this call. The application must ensure that the coordinates within the element are in the correct format for the presentation space (see GpiCreatePS).

GpiEndArea — End Area

This call ends the construction of a shaded area.

GpiEndArea (hps, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_NOT_IN_AREA
PMERR_COORDINATE_OVERFLOW

Remarks

The construction is started by the GpiBeginArea call. If necessary, a final line is constructed (to the starting point of the last figure) to close the area.

The current position is not changed, unless a closure line has to be drawn, in which case the current position is moved to the end point of the line.

Graphic Elements and Orders

Element Type: OCODE_GEAR

Order: End Area

GpiEndElement — End Element

SAA

This call terminates an element started by GpiBeginElement.

GpiEndElement (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_NOT_IN_ELEMENT

GpiEndPath — End Path

This call ends the specification of a path started by GpiBeginPath.

GpiEndPath (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_NOT_IN_PATH

Graphic Elements and Orders

Element Type: OCODE_GEPTH

Order: End Path

GpiEqualRegion — Equal Region

SAA

This call checks whether two regions are identical.

GpiEqualRegion (hps, Src1, Src2, *Equality*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The regions must be owned by the device identified by the currently associated device context.

Src1 *(HRGN)* — input
Handle of first region.

Src2 *(HRGN)* — input
Handle of second region.

Equality *(LONG)* — return
Equality/error indicator:

EQRGN_NOTEQUAL	Not equal
EQRGN_EQUAL	Equal
EQRGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION
PMERR_HRGN_BUSY

Remarks

Both regions must be of the same device class. It is invalid if the specified region is currently selected as the clip region (by GpiSetClipRegion).

GpiErase – Erase

This call clears the output display of the device context associated with the specified presentation space, to the reset color (CLR_BACKGROUND; see GpiSetColor).

GpiErase (hps, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

This operation is independent of the draw controls; see GpiSetDrawControl.

The call is subject to all clipping currently in force; that is, clip path, viewing limits, graphics field, clip region, and visible region.

This call does not perform any bounds collection, or correlation.

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiErrorSegmentData — Error Segment Data

SAA

This call returns information about the last error that occurred during a segment drawing operation.

GpiErrorSegmentData (hps, Segment, Context, *Off*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Segment *(LONG)* — output
Segment in which the error occurred.

Context *(LONG)* — output
Context of the error:

GPIE_SEGMENT	The error occurred while processing the contents of a retained segment.
GPIE_ELEMENT	The error occurred while processing the contents of a GpiElement call.
GPIE_DATA	The error occurred while processing the contents of a GpiPutData call.

Off *(LONG)* — return
Position.

This is either the byte offset or the element number, depending upon **Context**:

≥0	Position
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION

Remarks

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 these contexts:

- The error occurred within the data of a GpiPutData call
- The error occurred within the data of a GpiElement call.

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

GpiExcludeClipRectangle — Exclude Clip Rectangle

This call excludes a rectangle from the clipping region.

GpiExcludeClipRectangle (hps, Rectangle, *Complexity***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Rectangle *(RECT)* — input
Rectangle to be excluded.

The coordinates are world coordinates.

Complexity *(LONG)* — return
Complexity of clipping/error indicator.

The clipping complexity information includes the combined effects of:

- Clip path
- Viewing limits
- Graphics field
- Clip region
- Visible region (windowing considerations).

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_INV_RECT

Remarks

The boundaries of the rectangle are considered to be part of the interior, so that a point on the rectangle boundary is clipped (removed).

This call creates a clip region if one does not currently exist. The application is responsible for freeing this (with GpiDestroyRegion) if it subsequently selects another clip region (see GpiSetClipRegion). Any clip region still selected when the device context is closed is automatically freed.

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiFillPath — Fill Path

SAA

This call draws the interior of a path using the area attributes.

GpiFillPath (hps, Path, Options, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Path *(LONG)* — input
Identifier of path whose interior is to be drawn; it must be 1.

Options *(LONG)* — input
Fill option:

FPATH_ALTERNATE	The fill is done using the even/odd (alternate) rule; see GpiBeginArea.
FPATH_WINDING	The fill is done using the winding rule; see GpiBeginArea. This value must be selected if the path has been modified using GpiModifyPath.

The default is FPATH_ALTERNATE.

Hits *(LONG)* — return
Error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PATH_ID
PMERR_INV_FILL_PATH_OPTIONS
PMERR_PATH_UNKNOWN

Remarks

Any open figures within the path are closed.

The path is deleted when the interior has been drawn.

The boundaries of the area, as defined by the path, are considered to be part of the interior and are included in the fill.

If the current drawing mode (see GpiSetDrawingMode) is **draw** or **draw-and-retain**, the interior is drawn on the currently associated device. If the drawing mode is **retain**, this call is stored in the current segment, and output occurs when the segment is subsequently drawn in the usual way.

Graphic Elements and Orders

Element Type: OCODE_GFPTH

Note that GpiStrokePath also generates this element type.

Order: Fill Path

GpiFullArc — Full Arc

This call creates a full arc with its center at the current position.

GpiFullArc (hps, Control, Multiplier, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Control *(LONG)* — input
Interior/outline control.

Specifies whether the interior of the full arc should be filled, and whether the outline should be drawn:

DRO_FILL	Fill interior
DRO_OUTLINE	Draw outline
DRO_OUTLINEFILL	Draw outline and fill interior.

Multiplier *(ROF)* — input
Multiplier.

This determines the size of the arc, in relation to an arc with the current arc parameters. The implementation limit of the multiplier is 255.

The value must not be negative.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_ARC_CONTROL
PMERR_INV_MULTIPLIER

Remarks

The current position is not changed.

The arc parameters determine whether the full arc is drawn clockwise or counterclockwise.

Either the outline of the full arc, or its interior, or both, can be drawn.

If this call appears within an area or path definition, it generates a complete closed figure (DRO_OUTLINE must be specified). It must not occur within any other figure definition.

If correlation is in force, a hit always results if the pick aperture intersects the full arc boundary. However, if the pick aperture lies wholly within the figure, a hit only occurs if the interior is being drawn (DRO_FILL or DRO_OUTLINEFILL).

GpiFullArc — Full Arc

Graphic Elements and Orders

Element_Type: OCODE_GCFARC

Order: Begin Area

This order is generated only if **Control** is DRO_FILL or DRO_OUTLINEFILL.

Order: Full Arc at Current Position
Order: End Area

This order is generated only if **Control** is DRO_FILL or DRO_OUTLINEFILL.

GpiGetData — Get Data

This call retrieves a buffer of graphic data from the specified segment into the supplied buffer. The data is a list of drawing orders. For details of these, see Chapter 27, "Graphics Orders."

GpiGetData (hps, Segid, Offset, Format, Length, Data, *Count***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Segid *(LONG)* — input
Segment identifier.

Offset *(LONG)* — input/output
Segment offset.

A value used to indicate the position in the segment from which data is to be retrieved. It must be set to zero the first time GpiGetData is called. This indicates that data is to be obtained from the start of the segment. On return, it contains a value that can be used on a subsequent call to continue data retrieval.

The only possible values that can be specified are zero, or the value returned from a previous call.

Format *(LONG)* — input
Coordinate type required:

DFORM_NOCONV No coordinate conversion performed.

Length *(LONG)* — input
Length of data buffer.

Data *(BUFFER)* — output
Data buffer.

Count *(LONG)* — return
Length of returned data.

≥0 Number of bytes actually returned in **Data**
GPI_ALTERROR Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_SEG_OFFSET
PMERR_INV_GETDATA_CONTROL
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MICROPS_FUNCTION
PMERR_SEG_NOT_FOUND
PMERR_SEG_IS_CURRENT
PMERR_DATA_TOO_LONG

GpiGetData — Get Data

Remarks

If the buffer is large enough to contain the data requested, the data is returned and **Count** is set to show its length.

If the buffer is not large enough, the buffer is filled and **Count** is set to the length of the buffer. This may mean that there is an incomplete order at the end of the buffer; even so, it is possible to use GpiPutData subsequently, without having to scan the orders in the buffer.

The application can detect when it has been given all the data by checking the **Count** value. If this is less than the value of **Length** specified, there is no more data to be returned. If it is equal, there is more data, except in the case where the data just fits in the buffer, which is detected if another GpiGetData call is issued, and a **Count** of zero is returned.

No conversion of coordinates is performed for the DFORM_NOCONV value of the control parameter. The coordinates are in the presentation space format.

This call can be issued while there is a segment open, unless the open segment is the one referenced by this call, in which case it is an error.

The segment transform and viewing transform are not returned by this call.

GpiImage — Image

This call draws a rectangular image, with the top-left corner at the current position.

GpiImage (hps, Format, ImageSize, Length, Data, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Format *(LONG)* — input
Format of image data.

This is a reserved field; must be set to 0.

ImageSize *(SIZEROL)* — input
Size of image area (in pels).

The maximum width allowed is 2 040.

Length *(LONG)* — input
Length in bytes of image data.

Data *(BUFFER)* — input
Image data.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IMAGE_FORMAT
PMERR_INV_IMAGE_DATA_LENGTH
PMERR_INV_IMAGE_DIMENSION

Remarks

All images are a rectangular array of pels (display points), each pel being represented by one bit.

ImageSize, which defines the width and height of the image, determines how many pels there are in the horizontal and vertical directions.

Data determines which of the pels are visible; a '1' bit sets the associated pel, using the image (foreground) color and mix, and a '0' bit sets the pel using the image background color and mix.

The top left-hand corner of the image is placed at the current position, and the data supplied is drawn row by row, starting at the top. Each row is drawn from left to right and must be padded out to an integral number of bytes if the image width specified is not a multiple of 8. For example, if 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 exactly 2 bytes.

Within each byte the high-order bit is drawn on the left.

The length of image data specified must include the padding of each row of data. The length must be given in bytes, and an error message is issued if it is wrong.

Because of the different sizes of pels for different devices, the relationship of the image with respect to other graphics primitives is device-dependent.

The current position remains unchanged after the image has been drawn.

Graphic Elements and Orders

Element_Type: OCODE_GCBIMG

Order: Begin Image at Current Position
Order: Image Data

One order for each pel row of the image.

Order: End Image

GpiIntersectClipRectangle — Intersect Clip Rectangle

This call sets the new clip region to the intersection of the current clip region and the specified rectangle.

GpiIntersectClipRectangle (hps, Rectangle, *Complexity***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Rectangle *(RECT)* — input
Rectangle, the coordinates of which are world coordinates.

Complexity *(LONG)* — return
Complexity of clipping/error indicator.

The clipping complexity information includes the combined effects of:

- Clip path
- Viewing limits
- Graphics field
- Clip region
- Visible region (windowing considerations).

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_INV_RECT

Remarks

The boundaries of the rectangle are considered to be part of the interior, so that a point on the rectangle boundary is not clipped (removed) if it was previously within the clip region.

This call creates a clip region if one does not currently exist. The application is responsible for freeing this (with GpiDestroyRegion), if it subsequently selects another clip region (see GpiSetClipRegion). Any clip region still selected when the device context is closed is automatically freed.

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiLabel – Label

SAA

This call generates an element containing the specified label.

GpiLabel (hps, Label, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Label *(LONG)* – input
Required label.

No check is made on the value of this parameter.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_IN_ELEMENT

Remarks

This call has no effect unless a retained segment is being constructed. It is invalid within an element bracket. Duplicate labels within a segment are allowed.

Graphic Elements and Orders

Element Type: OCODE_GLABL

Order: Label

GpiLine — Line

This call draws a straight line from the current position to the specified end point.

GpiLine (hps, EndPoint, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

EndPoint *(POINT)* — input
End point of the line.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_INV_NESTED_FIGURES

Remarks

The current position is set to the end point of the line.

The line is drawn using the current values of the line color, line mix, line width, and line type attributes.

Graphic Elements and Orders

Element Type: OCODE_GCLINE

Note that GpiPolyLine also generates this element type.

Order: Line at Current Position

GpiLoadBitmap — Load Bit Map

SAA

This call creates and loads a bit map from a resource, and returns the bit-map handle.

GpiLoadBitmap (hps, Resource, Bitmap, Width, Height, *hbm*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The associated device should, if possible, hold the bit map in its own memory. Where this is not possible, main memory is used and the bit map is held in a format compatible with the device.

Resource *(RESID)* — input
Resource identity containing the bit map:

NULL Use the application's .EXE file.
Other Module handle returned from the OS/2 DosLoadModule call.

Bitmap *(IDENTITY)* — input
ID of the bit map within the resource file.

Width *(LONG)* — input
Width of the bit map in pels.

Height *(LONG)* — input
Height of the bit map in pels.

hbm *(HBITMAP)* — return
Bit-map handle:

≠0 Bit-map handle
GPI_ERROR Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_BITMAP_NOT_FOUND
PMERR_INV_BITMAP_DIMENSION

Remarks

Some bit-map calls, including drawing into the bit map, require it to be selected into a memory device context, using GpiSetBitmap. This is true whether device or main memory is used to hold the bit map.

The bit map is stretched to the specified **Width** and **Height**. If **Width** or **Height** is zero, the bit map is not stretched in that direction; when, for example, **Width** = 0, the bit map is not stretched horizontally, when **Height** = 0, it is not stretched vertically.

The bit map may have been created by the icon editor in bit-map mode.

There are a number of standard bit-map formats that should normally be adhered to. Other formats can be used if supported by the device.

The bit map is owned by the process from which this call is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

GpiLoadFonts — Load Fonts

This call loads one or more fonts from the specified resource file.

GpiLoadFonts (hab, Filename, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Filename *(STRL)* — input
Filename.

This is the fully-qualified name of the font resource. The file-name extension is ".FON."

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_FONT_FILE_DATA

Remarks

All of the fonts in the file become available for any presentation space (GPI or VIO) created by the same process. They are not available for any other process.

The format of the font definitions in the resource file is defined in Appendix C, "The Font-File Format."

When no longer required, the fonts may be unloaded with GpiUnloadFonts.

Note: Fonts loaded with GpiLoadFonts are not available for use for spooled printing, that is if a device type of OD_QUEUED is specified in DevOpenDC; in this case GpiCreateLogFont will never return FONT_MATCH for these fonts. To avoid this, install the fonts as public fonts by means of the control panel, on both the generating and the receiving workstations if these are different.

GpiLoadMetaFile — Load Metafile

SAA

This call loads data from a file into a metafile.

GpiLoadMetaFile (hab, Filename, *hmf*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Filename *(STRL)* — input
Filename.

The name of the file that is to be loaded into a metafile.

hmf *(HMF)* — return
Metafile handle or error:

≠0	Metafile handle
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_DOSOPEN_FAILURE
PMERR_DOSREAD_FAILURE

Remarks

A metafile is created, into which the data from the file is loaded. The handle of the metafile created is returned in **hmf**; it can be used on subsequent GpiPlayMetaFile or GpiDeleteMetaFile calls.

This call draws a marker with its center at the specified position.

GpiMarker (hps, Point, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — input
Position of the marker.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

The current position is moved to the specified point. The marker symbol that is drawn, is selected by the current values of the marker set and marker symbol attributes.

Graphic Elements and Orders

Element Type: OCODE_GMRK

Note that GpiPolyMarker also generates this element type.

Order: Marker at Given Position

GpiModifyPath — Modify Path

SAA

This call modifies the specified path.

GpiModifyPath (hps, Path, Mode, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Path *(LONG)* — input
Path identifier.

Identifier of the path to be modified; it must be 1.

Mode *(LONG)* — input
Modification required.

This must be:

MPATH_STROKE Convert the path to one describing the envelope of a wide line.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PATH_ID
PMERR_INV_MODIFY_PATH_MODE
PMERR_PATH_UNKNOWN
PMERR_COORDINATE_OVERFLOW

Remarks

This call converts the path to one describing the envelope of a wide line stroked using the current geometric wide-line attribute (see GpiSetLineWidthGeom). Note that this and GpiStrokePath are the only calls that can cause geometric wide lines to be constructed.

The envelope includes the effects of line joins, and line ends, according to the current values of these attributes (see GpiSetLineJoin and GpiSetLineEnd). Note these points:

- A line may be joined to an arc, for example. The common point is handled according to the line-join attribute, rather than applying line ends at each end.
- Any open figures within the path are not closed automatically.
- If a figure is closed using GpiCloseFigure, the joining rules are followed, rather than the ending rules, at the start and end point.
- The envelope takes account of any crossings, so that a character such as a stroked "X" does not have a hole in the middle when subsequently drawn in exclusive-OR mode.

After this call, the only calls that can be performed on the path are GpiFillPath, specifying the FPATH_WINDING option, or GpiSetClipPath, specifying the SCP_WINDING option.

Graphic Elements and Orders

Element Type: OCODE_GMPTH

Order: Modify Path

GpiMove — Move

SAA

This call moves the current position to the specified point.

```
GpiMove (hps, Point, Success)
```

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — input
Position to which to move.

This position is in world coordinates.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

This call 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 closed automatically if necessary.

This call is equivalent to the GpiSetCurrentPosition call, except that, if the current attribute mode is AM_PRESERVE (see GpiSetAttrMode), the current position is **not** saved before being set to a new value by the GpiMove call, and hence cannot be restored using the GpiPop call.

Graphic Elements and Orders

Element Type: OCODE_GSCP

Note that GpiSetCurrentPosition also generates this element type.

Order: Set Current Position

GpiOffsetClipRegion — Offset Clip Region

This call moves the clipping region by the specified displacement.

GpiOffsetClipRegion (hps, Point, *Complexity***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — input
Displacement.

The displacement by which the clipping region is to be moved, expressed as a relative point in world coordinates.

Complexity *(LONG)* — return
Complexity of clipping/error indicator.

The clipping complexity information includes the combined effects of:

- Clip path
- Viewing limits
- Graphics field
- Clip region
- Visible region (windowing considerations).

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_COORDINATE_OVERFLOW

Remarks

Note: This call must not be used when creating SAA-conforming metafiles (see "Metafile Restrictions" on page D-1).

GpiOffsetElementPointer – Offset Element Pointer

SAA

This call sets the element pointer, within the current segment, to the current value plus the specified offset.

GpiOffsetElementPointer (hps, offset, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

offset *(LONG)* – input
Offset to be added to the element pointer.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NOT_IN_RETAIN_MODE
PMERR_NO_CURRENT_SEG
PMERR_INV_IN_ELEMENT

Remarks

If the resulting value is negative, the element pointer is set to 0. If the resulting value is greater than the number of elements in the segment, it is set to the last element.

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress.

This call is invalid within an element bracket.

GpiOffsetRegion — Offset Region

This call moves a region.

GpiOffsetRegion (hps, Hrgn, Offset, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

The region must be owned by the device identified by the currently associated device context.

Hrgn *(HRGN)* — input
Region handle to be moved.

Offset *(POINT)* — input
Offset to be added to the region boundary.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_COORDINATE
PMERR_HRGN_BUSY

Remarks

This call moves the region to a new position. The new position is obtained by adding the value of **Offset** to all the points that define the region boundary.

It is invalid if the specified region is currently selected as the clip region (by GpiSetClipRegion).

GpiOpenSegment — Open Segment

SAA

This call opens a segment with the specified identification number.

GpiOpenSegment (hps, Segment, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Segment *(LONG)* — input
Segment identifier.

Must be zero or a positive number.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_MICROPS_FUNCTION
PMERR_ALREADY_IN_SEG
PMERR_PATH_INCOMPLETE (warning
PMERR_AREA_INCOMPLETE (warning
PMERR_INV_MODE_FOR_REOPEN_SEG
PMERR_DYNAMIC_SEG_ZERO_INV
PMERR_INV_MODE_FOR_OPEN_DYN
PMERR_UNCHAINED_SEG_ZERO_INV

Remarks

A segment is a way of grouping graphics primitives.

If the current drawing mode is **retain** or **draw-and-retain** (see GpiSetDrawingMode), the following occurs:

- If a nonzero identifier is given, and if a segment with the specified identifier does not already exist, a new retained segment is created. If one does already exist, it is reopened in **retain** mode (with the element pointer set to zero), but is an error in **draw-and-retain** mode.
- If an identifier of zero is given, a new retained segment is created, regardless of whether one with a zero identifier already exists. There can be more than one segment with an identifier of zero, but such segments can never subsequently be referenced by identifier. When they have been created, they continue to exist until all segments are deleted. Zero segments must be chained and can not be defined as dynamic.

If the current drawing mode is **draw**, a new nonretained segment is started. No check is made against any possible retained segment identifiers. The current attributes are set to default values (subject to the ATTR_FASTCHAIN segment attribute; see below).

The initial attributes of the segment are as set by GpiSetInitialSegmentAttrs. The attributes may subsequently be changed with GpiSetSegmentAttrs (except for a segment with an identifier of zero).

It is an error to try to open a new segment with a drawing mode of **draw** or **draw-and-retain**, with the ATTR_DYNAMIC segment attribute.

This call causes a segment bracket to be started. While the bracket is in effect, any primitive and attribute calls are considered to be part of the segment, and are stored in it if the drawing mode is **retain** or **draw-and-retain**. The bracket is terminated by a GpiCloseSegment. It is an error if GpiOpenSegment is issued when a segment is already open.

The following actions occur when drawing of a chained segment is started (either as it is passed across the API in **draw** or **draw-and-retain**, or as it is found during a draw operation):

- Current attributes and arc parameters are reset to default values.
- The current tag is reset to its default value.
- Current model transform is reset to unity.
- Current position is set to (0,0).
- The current clip path is set so as to cause no clipping.
- The current viewing limits are reset to their default values.
- The current viewing transform is set either to the value last set by GpiSetViewingTransformMatrix, or to the default value if no GpiSetViewingTransformMatrix call has been issued.

If the segment has the ATTR_FASTCHAIN attribute, the application should not depend upon whether or not these operations are performed. This avoids complications when interchanging picture data with other implementations.

Programming Note: The current clip region is not changed by this call.

If any primitive/attribute calls are issued immediately before this call (that is, outside a segment bracket), then any currently open area, path, or element brackets are terminated, as described for GpiCloseSegment, before the new segment is opened.

If the segment being defined is to be called from another segment (see GpiCallSegmentMatrix), ensure that the viewing transform (see GpiSetViewingTransformMatrix) is unity before first opening the segment.

The maximum number of retained segments allowed for a given presentation space at any time is 16 378.

GpiOutlinePath – Outline Path

This call draws the outline of a path.

GpiOutlinePath (hps, Path, Options, *Hits*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Path *(LONG)* – input
Identifier of path to be outlined; it must be 1.

Options *(LONG)* – input
Options:

Reserved; must be zero.

Hits *(LONG)* – return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PATH_ID
PMERR_INV_RESERVED_FIELD
PMERR_PATH_UNKNOWN

Remarks

The outline of the path is drawn, using the line attributes, including cosmetic line width (see GpiSetLineWidth) but not geometric line width (see GpiSetLineWidthGeom). This will normally have the same effect as if the lines, curves, etc. which comprise the path had been drawn without defining them as being within a path. However, if character strings (referencing outline fonts) are contained within the path, the outlines of the characters, without the interior fill, will be drawn by GpiOutlinePath, giving the appearance of hollow characters.

Open figures within the path are not closed automatically.

When the outline of the path has been drawn, the path is deleted.

Graphic Elements and Orders

Element Type: OCODE_GOPTH

Order: Outline Path

GpiPaintRegion — Paint Region

This call paints a region into a presentation space, using the current pattern attributes.

GpiPaintRegion (hps, hrgn, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

hrgn *(HRGN)* — input
Region handle.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_REGION_IS_CLIP_REGION
PMERR_INV_HRGN
PMERR_HRGN_BUSY

Remarks

The current GPI area foreground and background colors are used. Mixing is controlled by the area foreground mix only.

It is invalid if the specified region is currently selected as the clip region (by GpiSetClipRegion).

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiPartialArc — Partial Arc

SAA

This call draws a straight line, followed by an arc.

GpiPartialArc (hps, Center, Multiplier, StartAngle, SweepAngle, *Hits***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Center *(POINT)* — input
Center point.

Center of the arc.

Multiplier *(ROF)* — input
Multiplier.

This determines the size of the arc in relation to an arc with the current arc parameters.

The implementation limit for the multiplier is 255.

The value must **not** be negative.

StartAngle *(ROF)* — input
Start angle in degrees.

The value must be positive.

SweepAngle *(ROF)* — input
Sweep angle in degrees.

The value must be positive.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MULTIPLIER
PMERR_INV_COORDINATE
PMERR_INV_ANGLE_PARM
PMERR_INV_NESTED_FIGURES

Remarks

This call draws two figures:

- A straight line, from the current position to the starting point of an arc
- An arc, with its center at the specified point.

The full arc, of which the arc is a part, is identical to that defined by GpiFullArc. The part of the arc drawn by this primitive is defined by the parameters **StartAngle** and **SweepAngle**, that are the start and sweep angles, subtended from the center, if the current arc parameters specify a circular form. If they do not, these angles are skewed to the same degree that the ellipse is a skewed circle. **StartAngle** is measured counterclockwise from the x axis of the circle before application of the arc parameters. Both angles must be positive; whether the arc is drawn clockwise or counterclockwise is determined by the arc parameters.

Current position is updated to the final point on the arc.

Programming Note: This differs from GpiFullArc, where current position remains at the center of the figure. A primitive (such as GpiLine) following GpiPartialArc draws from the end point of the arc.

A segment of a pie can be drawn by the following call sequence:

1. GpiMove, to center of pie
2. GpiPartialArc, drawing one spoke and the arc
3. GpiLine, back to center.

The third step can be performed implicitly by 'autoclosure' if an area is being drawn.

A closed figure bounded by a chord and an arc can be drawn by the following call sequence:

1. GpiSetLineType to invisible.
2. GpiPartialArc, with **StartAngle** = angle2, and **SweepAngle** = 0, to define one end of the chord.
3. GpiSetLineType to visible.
4. GpiPartialArc, with **StartAngle** = angle1, and **SweepAngle** = angle2 − angle1.

(In the second example, angle2 is greater than angle1. If the interior of the chord is to be shaded, the area must start after step 2 or 3.)

A sweep angle of greater than 360 degrees is valid, and means that after the initial line a full arc is drawn, followed by a partial arc with a sweep angle of (**SweepAngle** MOD 360) degrees.

Graphic Elements and Orders

Element Type: OCODE_GCPARC

Order: Partial Arc at Current Position

GpiPlayMetaFile — Play Metafile

SAA

This call causes a metafile to be "played" into a presentation space.

GpiPlayMetaFile (hps, hmf, Count1, Optarray, SegCount, Count2, Desc, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

hmf *(HMF)* — input
Metafile handle.

Handle of the metafile containing the data.

Count1 *(LONG)* — input
Count of elements in **Optarray**.

Optarray *(LONG*Count1)* — input
Array of options for playing.

The values of the elements in this array determine what action is to be taken when the metafile is played into the specified presentation space. The elements in the array are numbered consecutively, starting with PMF_SEGBASE. The element number constants start with 0 or with 1, depending on the language; refer to the appropriate bindings reference. Any elements in the array that are not set to one of the values defined below must be set to zero.

Optarray.[PMF_SEGBASE]
Reserved; must be zero.

Optarray.[PMF_LOADTYPE]
Specifies what transformations should be performed on the imported picture. The options are:

LT_DEFAULT
The default; same as LT_NOMODIFY

LT_NOMODIFY
The graphics are restored using the current viewing transform (see GpiSetViewingTransformMatrix), rather than the ones that were in use when the data was created. This is the default action.

Any change to the graphics field or default viewing transform during the course of the picture will be ignored if this option is specified (or defaulted).

LT_ORIGINALVIEW
The graphics are restored using the viewing transforms that are in the metafile.

The default viewing transform of the presentation space is not altered (unless RES_RESET is specified). However, any changes to the default viewing transform that occur during the course of the picture (and also any graphics field clipping) cause changes to the values in the presentation space.

Optarray.[PMF_RESOLVE]
Reserved; must be zero.

Optarray.[PMF_LCIDS]
Specifies the action to be taken for any logical font definitions, or bit maps referenced by local identifiers for use as shading patterns that are held in the metafile.

The options are:

LC_DEFAULT
Default; same as LC_NOLOAD.

LC_NOLOAD
Do not load such objects. This is the default, and is used where the application expects the correct objects to be already loaded.

LC_LOADDISC
Load all objects referenced in the metafile, first deleting any already existing in the presentation space, for which the referenced local identifier is already in use.

Optarray.[PMF_RESET]

Specifies whether the presentation space should be reset before playing the metafile, with the page units and size being set as defined in the metafile.

The options are:

RES_DEFAULT
Default; same as RES_NORESET.

RES_NORESET
Do not perform a reset.

RES_RESET
Reset the presentation space, before loading any logical fonts, color tables, segments, and so on, as follows:

1. Reset the page units and page size to the values contained in the metafile.
2. Set up default transformations, based on the page units and size, as if the presentation space had just been created with these values.
3. Further modify the device transform to ensure that the physical size of the metafile picture is preserved. (Only performed if the page units in the metafile are not PU_ARBITRARY or PU_PELS.)
4. Perform the equivalent of GpiResetPS (option GRES_ALL).
5. Set the default viewing transform to the value specified in the metafile.

This option should normally be used with the PMF_LOADTYPE option of LT_ORIGINALVIEW and LC_LOADDISC, but this is not enforced.

Optarray.[PMF_SUPPRESS]

Specifies whether the playing of this metafile actually occurs. This is provided to allow an application to use the PMF_RESET option, and then to regain control to perform further presentation space modifications if necessary, before playing the remainder of the metafile.

The options are:

SUP_DEFAULT
Default; same as SUP_NOSUPPRESS.

SUP_NOSUPPRESS
Do not suppress the remainder of the metafile.

SUP_SUPPRESS
Suppress the remainder of the metafile.

If this option is selected, only processing as determined by the PMF_RESET option is performed. The remainder of the metafile, and all other options, are ignored.

Optarray.[PMF_COLORTABLES] Specifies the action to be taken with respect to any color table implied or present within the metafile.

The options are:

CTAB_DEFAULT
Default; same as CTAB_NOMODIFY.

CTAB_NOMODIFY
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.

CTAB_REPLACE
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.

Optarray.[PMF_COLORREALIZABLE] Specifies whether the color table data contained in the metafile should be loaded with the LCOL_REALIZABLE option or not (see GpiCreateLogColorTable).

The options are:

CREA_DEFAULT
Default; same as CREA_NOREALIZE

CREA_REALIZE
Load the color table with the realizable option set. This is used where the application playing the metafile requires that the color table be realized.

CREA_NOREALIZE
Load the color table with the realizable option off. This is the default.

Optarray.[PMF_DEFAULTS] Specifies how the drawing defaults contained in the metafile should be used (see GpiSetDefAttrs, GpiSetDefViewingLimits, GpiSetDefTag, and GpiSetDefArcParams).

The options are:

DDEF_DEFAULT
Default; same as DDEF_IGNORE

DDEF_IGNORE
Ignore any drawing default values in the metafile.

DDEF_LOADDISC
Change any drawing default values in the presentation space that are specified in the metafile, to the values contained in the metafile.

SegCount *(LONG)* — output
Reserved.

The value 0 is always returned.

Count2 *(LONG)* — input
Count of bytes in **Desc**.

Desc *(STRL)* — output
Descriptive record.

Desc is a buffer that, on return, contains the descriptive record, of up to 253 bytes, that is saved when the metafile is created (see DevOpenDC). This is null-terminated, even if it has to be truncated.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HMF
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_PLAY_METAFILE_OPTION
PMERR_INCOMPATIBLE_METAFILE
PMERR_INV_METAFILE
PMERR_INV_MICROPS_ORDER
PMERR_STOP_DRAW_OCCURRED (warning
PMERR_INV_OUTSIDE_DRAW_MODE
PMERR_INV_ELEMENT_POINTER
PMERR_INV_IN_CURRENT_EDIT_MODE
PMERR_PROLOG_ERROR
PMERR_DUP_SEG
DPC errors
Reset Presentation Space errors; see GpiResetPS
Set Default View Matrix errors; see GpiSetDefaultViewMatrix
Set Code Page Errors; see GpiSetCp
Create Log Font errors; see GpiCreateLogFont
Set Bit Map Id errors; see GpiSetBitmapId
Create Log Color Table errors; see GpiCreateLogColorTable
Set Clip Region errors; see GpiSetClipRegion
Close Segment errors; see GpiCloseSegment
Open Segment errors; see GpiOpenSegment
Delete Set Id errors; see GpiDeleteSetId
Erase errors; see GpiErase
Intersect Clip Rectangle errors; see GpiIntersectClipRectangle
Exclude Clip Rectangle errors; see GpiExcludeClipRectangle
Offset Clip Region errors; see GpiOffsetClipRegion
Paint Region errors; see GpiPaintRegion
Create Region errors; see GpiCreateRegion
Create Bit Map errors; see GpiCreateBitmap.

Remarks

Whether the graphics are drawn, or retained in segment store, or both, depends upon the current drawing mode (see GpiSetDrawingMode) in the presentation space, for the chained and unchained segment contexts, as appropriate. If chained segments are retained, they are added to the end of any existing segment chain. An error is raised if a segment is to be retained, and it has the same (nonzero) identifier as a currently existing segment.

A segment must not be open when this call is issued. At the completion of the call, there is no open segment.

The application may need to reset the presentation space by GpiResetPS, before issuing this call. Alternatively, the PMR_RESET option on this call may be suitable.

Segments retain the segment attributes that they originally possessed.

GpiPointArc — Point Arc

This call creates an arc, using the current arc parameters, through three points, starting at the current position.

GpiPointArc (hps, Points, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Points *(POINT*2)* — input
Intermediate and end points.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_INV_NESTED_FIGURES

Remarks

The first element of the **Points** array defines an intermediate point along the arc, and the second element identifies the end point of the arc. Upon completion, current position is set to the end point of the arc.

Graphic Elements and Orders

Element Type: OCODE_GCARC

Order: Arc at Current Position

GpiPolyFillet — Polyfillet

SAA

This call draws a curve starting at the current position and defined by the points supplied.

GpiPolyFillet (hps, Count, Points, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of points.

Must not be negative. Zero is valid but causes no output.

Points *(POINT*Count)* — input
Array of points.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORDINATE
PMERR_INV_NESTED_FIGURES

Remarks

If two points are supplied, an imaginary straight line is drawn from the current position to the first point and a second straight line from the first point to the second. A curve is then constructed, starting at the current position and tangential to the first straight line. The curve is drawn such that it reaches the last point at a tangent to the second straight line. Figure 4-1 on page 4-117 shows the curve constructed, given current position A and the two points B and C.

If more than two points are supplied, a series of imaginary straight lines is constructed through them (as in the GpiPolyLine call). All of the straight lines except the first and last are then divided in two at their mid-points. A series of curved fillets is then drawn, each starting at the end point of the last, at one of the mid-points. Figure 4-2 on page 4-117 shows the curve constructed, given current position A and three points B, C, and D.

The current position is set to the last point.

Each individual fillet always lies within the area bounded by the start, end, and control points.

It is not an error for any of the points to be coincident.

The maximum number of fillets allowed in the polyfillet is more than 4 000.

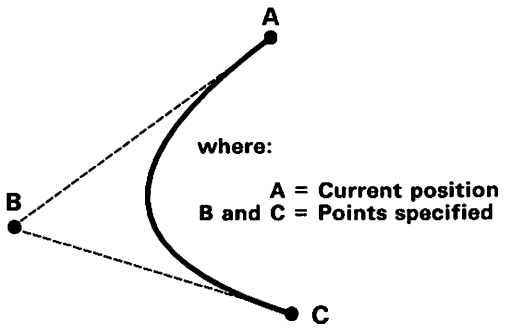


Figure 4-1. GpiPolyFillet Example A

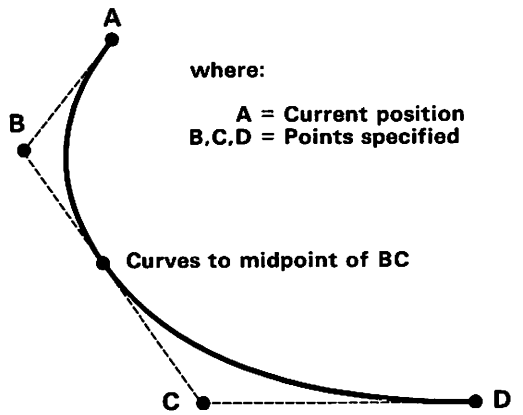


Figure 4-2. GpiPolyFillet Example B

Graphic Elements and Orders

Element Type: OCODE_GCFLT

Order: Fillet at Current Position

As many of these orders are generated as is necessary to hold the specified fillets.

GpiPolyFilletSharp — Polyfillet Sharp

SAA

This call creates a fillet on a series of connected lines, with the first line starting at the current position. Subsequent points identify the end points of the lines.

GpiPolyFilletSharp (hps, Count, Points, Sharpness, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Count of points.

This is the number of points specified in **Points**. It must be 2*f, where f is the number of fillets; the value must be a positive even number. Zero is valid but causes no output.

Points *(POINT*Count)* — input
An array of points.

These points are set as follows:

```
c1, e1, c2, e2, c3, e3, ... cf, ef
```

where:

cf is the control point for the f'th fillet
ef is the end point of the f'th fillet.

Sharpness *(ROF*Count/2)* — input
Array of sharpness values.

These give the sharpness of successive fillets.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK Successful
GPI_HITS Correlate hit(s)
GPI_ERROR Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORDINATE
PMERR_INV_SHARPNESS_PARM
PMERR_INV_NESTED_FIGURES

Remarks

The first fillet is drawn using the two imaginary lines, one from current position to its control point (the first point specified in **Points**), and one from this point to the second point specified in **Points**. The fillet starts from current position, and ends at this second point. It is tangential to the first line at current position, and to the second line at the second point of **Points**. The sharpness of this fillet is given by the first element of the **Sharpness** array.

Each subsequent fillet is drawn starting from the end point of the previous fillet, and uses the next two lines in the sequence, in a similar way. Therefore, two points and one sharpness value are required for each fillet.

The differences from GpiPolyFillet are:

- The sharpness of each fillet is explicitly specified.
- Both the control and the end point of each fillet are explicitly specified.
- Adjacent fillets, generally, have a discontinuity in gradient, unless the points are chosen so that this is not the case.

The sharpness of each fillet is defined as follows. Let A and C be the start and end points, respectively, of the fillet, and let B be the control point (see Figure 4-3). Let W be the mid-point of AC. Let D be the point where the fillet intersects WB.

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.

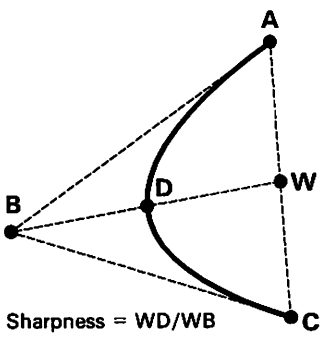


Figure 4-3. GpiPolyFilletSharp Example

On completion, the current position is the end point of the last line in the series. Each individual fillet always lies within the area bounded by the start, end, and control points.

It is not an error for any of the points to be coincident.

The maximum number of fillets allowed is more than 2 000.

Graphic Elements and Orders

Element Type: OCODE_GCSFLT

Order: Sharp Fillet at Current Position

As many of these orders are generated as is necessary to hold the specified fillets.

GpiPolyLine — Polyline

SAA

This call draws a series of straight lines starting at the current position and connecting the points specified.

GpiPolyLine (hps, Count, Points, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of points

Must not be negative. Zero is valid but causes no output.

Points *(POINT*Count)* — input
Array of points.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORDINATE
PMERR_INV_NESTED_FIGURES

Remarks

On completion, current position is set to the last point.

The maximum number of lines allowed in the polyline is more than 8 000.

Graphic Elements and Orders

Element Type: OCODE_GCLINE

Note that GpiLine also generates this element type.

Order: Line at Current Position

As many of these orders are generated as is necessary to hold the specified points.

GpiPolyMarker — Polymarker

This call draws a marker, selected by the current values of the marker set and marker symbol attributes, at each of the specified positions.

GpiPolyMarker (hps, Count, Points, *Hits***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of points.

Must not be negative. Zero is valid but causes no output.

Points *(POINT*Count)* — input
Array of points.

A marker is drawn at each of these points.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORDINATE

Remarks

The center of each marker is drawn at the specified position(s).

On completion, the current position is set to the position of the last marker in the series.

Graphic Elements and Orders

Element Type: OCODE_GMRK

Note that GpiMarker also generates this element type.

Order: Marker at Given Position

As many of these orders are generated as is necessary to hold the specified points.

GpiPolySpline — Polyspline

SAA

This call creates a succession of Bézier splines.

GpiPolySpline (hps, Count, Points, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Count of points.

This is the number of points specified in **Points**. It must be 3*s, where s is the number of splines; the value must not be negative, and it must be divisible by 3. Zero is valid but causes no output.

Points *(POINT*Count)* — input
An array of points.

The points are given in this order:

```
c11, c12, e1, c21, c22, e2, ... cs1, cs2, es
```

where:

cs1 is the first control point of the s'th spline
cs2 is the second control point of the s'th spline
es is the end point of the s'th spline.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK Successful
GPI_HITS Correlate hit(s)
GPI_ERROR Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORDINATE
PMERR_INV_NESTED_FIGURES

Remarks

The first Bézier spline starts from the 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.

On completion, the current position is set to the last specified point. Each individual spline always lies within the area bounded by the start, end, and control points.

It is not an error for any of the points to be coincident.

The maximum number of splines allowed is more than 2 500.

Graphic Elements and Orders

Element Type: OCODE_GCBEZ

Order: Bézier Spline at Current Position

As many of these orders are generated as is necessary to hold the specified splines.

GpiPop — Pop

SAA

This call restores the primitive attributes that have been saved on the stack.

GpiPop (hps, Count, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of attributes to be restored.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_LENGTH_OR_COUNT
PMERR_SEG_CALL_STACK_EMPTY

Remarks

Each time a primitive attribute call (such as color, or line type) is issued and the attribute mode is set to AM_PRESERVE, the values are put into a "Last in, First out" stack.

This call can reset the current attribute values (starting with the last one set) to the previous value; this is known as "popping." This allows a called segment to change the values of the attributes, and allows them to be restored on return to the caller (an implicit GpiPop call is performed for each preserved attribute when returning from a called segment).

When inside an area or path definition, this call is only valid if the attribute being popped is valid inside an area or path definition.

Programming Note: It is not possible to check whether the attribute to be popped is valid before issuing this call.

Graphic Elements and Orders

Element Type: OCODE_GPOP

Order: Pop

Count of these orders are generated.

GpiPtInRegion — Point In Region

This call checks whether a point lies within a region.

GpiPtInRegion (hps, hrgn, Point, *Inside*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The region must be owned by the device identified by the currently associated device context.

hrgn *(HRGN)* — input
Region handle.

Point *(POINT)* — input
Point to be checked.

The point is in device coordinates.

Inside *(LONG)* — return
Inside/error indicator:

PRGN_OUTSIDE	Not in region
PRGN_INSIDE	In region
PRGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_INV_COORDINATE
PMERR_REGION_IS_CLIP_REGION
PMERR_HRGN_BUSY

Remarks

It is invalid if the specified region is currently selected as the clip region (by GpiSetClipRegion).

GpiPtVisible — Point Visible

SAA

This call checks whether a point is visible within the clipping region of the device associated with the specified presentation space.

GpiPtVisible (hps, Point, *Visibility***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — input
Point to be checked.

The point is given in world coordinates.

Visibility *(LONG)* — return
Visibility indicator:

PVIS_INVISIBLE	Not visible
PVIS_VISIBLE	Visible
PVIS_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

For the purposes of this call, the clipping region is defined as the intersection between the application clipping region, and any other clipping, including windowing.

GpiPutData — Put Data

This call passes a buffer of graphics orders to the current segment, or draws the orders, or both of these. For details of the orders, see Chapter 27, "Graphics Orders."

GpiPutData (hps, Format, Length, Data, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Format *(LONG)* — input
Coordinate type used:

DFORM_NOCONV	No coordinate conversion performed
DFORM_S370SHORT	S/370 format short (2-byte) integers
DFORM_PCSHORT	PC format short (2-byte) integers
DFORM_PCLONG	PC format long (4-byte) integers.

Length *(LONG)* — input/output
Length of graphic data.

Set by the application to the length of order data in **Data**. If an incomplete order occurred, it is updated, on return, to the offset of the start of the incomplete order.

Length must not be greater than 63 KB.

Data *(BUFFER)* — input
Orders to be copied.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PUTDATA_FORMAT
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MICROPS_FUNCTION
PMERR_DATA_TOO_LONG
PMERR_INV_ELEMENT_POINTER
PMERR_INV_REPLACE_MODE_FUNC
PMERR_ORDER_TOO_BIG
DPC errors

Remarks

The orders passed may be added to the current segment, drawn immediately, or both, depending on the current drawing mode (see GpiSetDrawingMode), and whether the primitives are within a segment.

If there is an incomplete order at the end of the buffer, **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.

The orders End Prolog and Set Viewing Transform are not allowed.

This call is valid within an element bracket (see GpiBeginElement). It can contain GpiBeginElement and GpiEndElement orders, while these are in the correct sequence with respect to the currently opened segment in segment store.

The data in the buffer is converted, if necessary, to the presentation space format (defined when the presentation space is first created; see GpiCreatePS).

This call is invalid if the editing mode (see GpiSetEditMode) is set to SEGEM_REPLACE, and also in SEGEM_INSERT mode if the element pointer is not pointing to the last element.

GpiQueryArcParams — Query Arc Parameters

This call returns the current arc parameters used to draw full, partial, and 3-point arcs.

GpiQueryArcParams (hps, ArcParams, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

ArcParams *(ARCPARAM)* — output
Arc parameters.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

Arc parameters are set by GpiSetArcParams.

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryAttrMode — Query Attribute Mode

SAA

This call returns the current value of the attribute mode, as set by GpiSetAttrMode.

GpiQueryAttrMode (hps, *Mode*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Mode *(LONG)* — return
Current attribute mode:

≥0	Current attribute mode
AM_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

GpiQueryAttrs — Query Attributes

This call returns current attributes for the specified primitive type.

GpiQueryAttrs (hps, PrimType, AttrMask, Attrs, *DefMask***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

PrimType *(LONG)* — input
Primitive type.

This is the type of primitive for which attributes are to be queried, as follows:

PRIM_LINE	Line and arc primitives
PRIM_CHAR	Character primitives
PRIM_MARKER	Marker primitives
PRIM_AREA	Area primitives
PRIM_IMAGE	Image primitives.

AttrMask *(BIT32)* — input
Attributes mask.

Each flag that is set indicates that the corresponding flag in **DefMask** is to be updated, and that if the corresponding attribute is not currently set to default, its value is to be returned in the **Attrs** buffer.

If all flags in **AttrMask** are zero, the **Attrs** buffer address is not used.

Attrs *(BUNDLE)* — output
Attributes.

Attrs is a buffer in which is returned the value of each non-default attribute for which the **AttrMask** flag is set, in the order specified in GpiSetAttrs for the particular primitive type.

Only data for attributes for which the appropriate flag in **AttrMask** is set is updated, so **Attrs** need only be large enough for the highest offset attribute to be returned (see GpiSetAttrs).

The data returned in **Attrs** for any attribute for which the **AttrMask** flag is set, but which is currently set to default, is undefined.

DefMask *(LONG)* — return
Defaults mask.

As **DefMask** in GpiSetAttrs:

GPI_ALTERROR	Error occurred
Positive	Defaults mask, numeric value can be greater than or equal to zero.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PRIMITIVE_TYPE
PMERR_UNSUPPORTED_ATTR
PMERR_INV_IN_RETAIN_MODE

GpiQueryAttrs – Query Attributes

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**. This call returns a mask, similar in meaning to **DefMask** in GpiSetAttrs. Each flag in the returned mask is updated if the corresponding flag in **AttrMask** is set. It is set if the attribute is set to the default, otherwise it is reset. Other flags are undefined.

The parameters returned by this call may be used to reinstate exactly the same attributes as are queried, using GpiSetAttrs.

GpiQueryBackColor — Query Background Color

This call returns the current value of the (character) background color attribute, as set by the GpiSetBackColor call.

GpiQueryBackColor (hps, *Color*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Color *(LONG)* — return
Background color:

CLR_ERROR	Error
CLR_DEFAULT	Default
Otherwise	Background color index.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryBackMix — Query Background Mix

SAA

This call returns the current value of the (character) background color-mixing mode, as set by the GpiSetBackMix call.

GpiQueryBackMix (hps, *MixMode*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

MixMode *(LONG)* — return
Background mix:

BM_DEFAULT	Default
>0	Background mix mode
BM_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryBitmapBits — Query Bit-Map Bits

This call transfers data from a bit map to application storage.

GpiQueryBitmapBits (hps, ScanStart, Scans, Buffer, InfoTable, *ScansReturned***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

ScanStart *(LONG)* — input
Starting line number.

Scan-line number at which the data transfer is to start, counting from zero as the bottom line.

Scans *(LONG)* — input
Number of scan lines to be returned.

Buffer *(BUFFER)* — output
Data area.

Data area into which the bit-map data is copied.

InfoTable *(BITMAPINFO)* — input/output
Bit-map information table.

ScansReturned *(LONG)* — return
Number of scan lines actually returned:

≥0	Number of scan lines actually returned
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_INFO_TABLE
PMERR_NO_BITMAP_SELECTED
PMERR_INV_SCAN_START
PMERR_INCORRECT_DC_TYPE

Remarks

The presentation space must be currently associated with a memory device context, which has a bit map currently selected.

The **InfoTable** must be initialized by the application with the values of **planes** and **bitcount**, set to the format required. The standard bit-map formats are supported, plus any known to be supported by the device (see GpiQueryDeviceBitmapFormats). This call returns the values of **bitmapwidth**, **bitmapheight** and the **color** array filled in by the system.

The bit-map data is converted where necessary.

Buffer must point to a storage area large enough to contain data for the requested number of scan lines. The amount of storage required for one scan line can be determined by GpiQueryBitmapParameters. It is

```
((bitcount*bitmapwidth + 31)/32)*planes*4 bytes
```

The storage required for the entire bit map is this value multiplied by **bitmapheight**.

GpiQueryBitmapDimension – Query Bit-Map Dimension

This call returns the width and height of a bit map, as specified on a previous GpiSetBitmapDimension call.

GpiQueryBitmapDimension (hbm, BitmapDimension, *Success***)**

Parameters

hbm *(HBITMAP)* – input
Bit-map handle.

BitmapDimension *(SIZEROL)* – output
Size of bit map.

The width and height of the bit map in 0.1 millimeter units.

If not set by GpiSetBitmapDimension, zeros are returned.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HBITMAP
PMERR_HBITMAP_BUSY

GpiQueryBitmapHandle — Query Bit-Map Handle

This call returns the handle of the bit map currently tagged with the specified local identifier (lcid).

GpiQueryBitmapHandle (hps, Lcid, *hbm*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Lcid *(LONG)* — input
Local identifier.

hbm *(HBITMAP)* — return
Bit-map handle:

≠0	Bit-map handle
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SETID
PMERR_SETID_HAS_NO_BITMAP

Remarks

An error is raised if a bit map is not currently tagged with the specified lcid.

GpiQueryBitmapParameters — Query Bit-Map Parameters

SAA

This call returns information about a bit map identified by the bit-map handle.

GpiQueryBitmapParameters (hbm, Data, *Success***)**

Parameters

hbm *(HBITMAP)* — input
Bit-map handle.

Data *(BITMAPINFOHEADER)* — output
Bit-map information header.

This is a structure, that on return, is filled with data for the specified bit map. The structure includes the elements (width, height, planes, bitcount) of a bit-map information table.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HBITMAP
PMERR_HBITMAP_BUSY

GpiQueryBoundaryData – Query Boundary Data

This call returns the boundary data.

GpiQueryBoundaryData (hps, Boundary, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Boundary *(RECT)* – output
Boundary data.

A rectangle structure in which the boundary data is returned, containing the following fields:

xmin Lowest x value found
ymin Lowest y value found
xmax Highest x value found
ymax Highest y value found.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_COORDINATE_OVERFLOW

Remarks

This call returns the boundary data set upon completion of the last boundary calculation. Boundary data is returned as the coordinates in model space.

Boundary data is inclusive. A null boundary is indicated if the 'lowest' value or either x or y is greater than the 'highest'. After GpiResetBoundaryData, the 'lowest' values are the maximum positive numbers, and the 'highest' values the maximum negative numbers.

GpiQueryCharAngle — Query Character Angle

SAA

This call returns the current value of the character baseline angle.

GpiQueryCharAngle (hps, Angle, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Angle *(GRADIENT)* — output
Baseline angle.

A point, relative to (0,0), that defines the character baseline angle vector.

If the character angle is currently set to the default value, (0,0) is returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryCharBox — Query Character Box

This call returns the current value of the character box attribute, as set by the GpiSetCharBox call.

GpiQueryCharBox (hps, Size, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Size *(SIZEROF)* — output
Character-box size.

If the character box is currently set to the default, the default size is returned. This is the size returned by DevQueryCaps (CAPS_GRAPHICS_CHAR_WIDTH and CAPS_GRAPHICS_CHAR_HEIGHT), converted to presentation page space.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

In general, this call does not return the same box as GpiQueryTextBox for an average-sized character. For outline fonts the character-box attribute is mapped to a particular font dimension related to the point size, for raster fonts it does not correspond to any font metric (see GpiSetCharMode).

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryCharDirection — Query Character Direction

SAA

This call returns the current value of the character direction attribute, as set by the GpiSetCharDirection call.

GpiQueryCharDirection (hps, *Direction***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Direction *(LONG)* — return
Character direction:

CHDIRN_DEFAULT	Default
>0	Character direction
CHDIRN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryCharMode — Query Character Mode

This call returns the current value of the character-mode attribute, as set by the GpiSetCharMode call.

GpiQueryCharMode (hps, *Mode*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Mode *(LONG)* — return
Character mode:

CM_DEFAULT	Default
>0	Character mode
CM_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryCharSet — Query Character Set

SAA

This call returns the character-set local identifier (lcid), as set by the GpiSetCharSet call.

GpiQueryCharSet (hps, *Lcid***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Lcid *(LONG)* — return
Character-set local identifier:

LCID_DEFAULT	Default
>0	Local identifier
LCID_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryCharShear — Query Character Shear

This call returns the value of the current character-shear angle, as set by the GpiSetCharShear call.

GpiQueryCharShear (hps, Shear, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Shear *(POINT)* — output
Character shear.

A point, relative to (0,0), that defines the character shear vector.

If the character shear is currently set to the default, (0,1) is returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryCharStringPos — Query Character String Positions

SAA

This call processes a string as if it is being drawn under the current character attributes using GpiCharStringPos, and returns the positions in the string at which each character would be drawn.

GpiQueryCharStringPos (hps, Options, Count, String, Xincrements, Positions, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(BIT32)* — input
Option flag:

CHS_VECTOR Increments vector supplied (**Xincrements**). If zero, **Xincrements** is ignored.

Count *(LONG)* — input
Length of the string.

String *(STR)* — input
The character string to be examined.

Xincrements *(ROL****Count)*** — input
Vector of x increment values.

These are signed values in world coordinates. This parameter is ignored if CHS_VECTOR is not set.

Positions *(POINT****Count**+*1)* — output
Array of points.

The positions of each character in world coordinates. The first point returned is the initial current position, and the last point is the new current position if the string has been drawn.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE
PMERR_INV_CHAR_POS_OPTIONS
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORDINATE
PMERR_COORDINATE_OVERFLOW

Remarks

A vector of increments can be specified, allowing control over the positioning of each character after the first. These are distances measured in world coordinates (along the baseline for left-to-right and right-to-left character directions, and along the shearline for top-to-bottom and bottom-to-top). The i'th increment is the distance of the reference point 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.

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryCharStringPosAt — Query Character String Positions At

This call processes a string as if it is being drawn under the current character attributes using GpiCharStringPosAt, and returns the positions in the string at which each character would be drawn.

GpiQueryCharStringPosAt (hps, Start, Options, Count, String, Xincrements, Positions, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Start *(POINT)* — input
Starting position.

Options *(BIT32)* — input
Option flags:

CHS_VECTOR Increments vector supplied (**Xincrements**). If zero, **Xincrements** is ignored.

Count *(LONG)* — input
Length of the string.

String *(STR)* — input
Character string to be examined.

Xincrements *(ROL****Count)*** — input
Vector of x increment values.

These are signed values in world coordinates. This parameter is ignored if CHS_VECTOR is not set.

Positions *(POINT****Count**+*1)* — output
Array of points, in which the positions of each character in world coordinates are returned.

The first point returned is the initial current position, and the last point is the new current position if the string has been drawn.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE
PMERR_INV_CHAR_POS_OPTIONS
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COORDINATE
PMERR_COORDINATE_OVERFLOW

GpiQueryCharStringPosAt — Query Character String Positions At

SAA

Remarks

A vector of increments can be specified, allowing control over the positioning of each character after the first. These are distances measured in world coordinates (along the baseline for left-to-right and right-to-left character directions, and along the shearline for top-to-bottom and bottom-to-top). The i'th increment is the distance of the reference point 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.

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryClipBox — Query Clip Box

This call returns the dimensions of the tightest rectangle which completely encloses the intersection of **all** the clipping definitions.

GpiQueryClipBox (hps, Bound, *Complexity***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Bound *(RECT)* — output
Bounding rectangle.

The coordinates of the bounding rectangle, in world coordinates.

Complexity *(LONG)* — return
Complexity/error indicator:

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_COORDINATE_OVERFLOW

Remarks

The clipping definitions include the combined effects of:

- Clip path
- Viewing limits
- Graphics field
- Clip region
- Visible region (windowing considerations).

Points on the borders of the rectangle returned are considered to be included within the rectangle. If the intersection is null, the rectangle returned has the right boundary less than the left, and the top boundary less than the bottom.

GpiQueryClipRegion – Query Clip Region

This call returns the handle of the currently selected clip region.

GpiQueryClipRegion (hps, *hrgn*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

hrgn *(HRGN)* – return
Clip-region handle (if any):

NULL	Null handle (no region is selected)
HRGN_ERROR	Error
Otherwise	Clip region handle.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

If there is no currently selected clip region, a null handle is returned.

GpiQueryColor — Query Color

This call returns the current value of the (character) color attribute, as set by the GpiSetColor call.

GpiQueryColor (hps, *Color***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Color *(LONG)* — return
Color attribute:

CLR_ERROR	Error
CLR_DEFAULT	Default
Otherwise	Color index.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryColorData – Query Color Data

SAA

This call returns information about the current logical color table.

GpiQueryColorData (hps, Count, Array, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Count *(LONG)* – input
Number of elements.

Number of elements supplied in **Array**.

Array *(LONG*Count)* – output
Array.

On return this array contains:

Array[QCD_LCT_FORMAT]	Format of loaded color table if any: **LCOLF_DEFAULT** Default color table is in force. **LCOLF_INDRGB** Color table loaded which provides translation from index to RGB. **LCOLF_RGB** Color index = RGB.
Array[QCD_LCT_LOINDEX]	Smallest color index in the color table; always zero.
Array[QCD_LCT_HIINDEX]	Largest color index in the color table; never less than 15.

The array elements are numbered consecutively, starting with Array[QCD_LCT_FORMAT]. The element number constants start with either 0 or 1, depending upon the language (see the appropriate bindings reference).

Information is only returned for the number of elements supplied. Any extra elements supplied, beyond the three described above, are set to zero by the system.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT

GpiQueryColorIndex — Query Color Index

This call returns the color index of the device color that is closest to the specified RGB color representation for the device connected to the specified presentation space.

GpiQueryColorIndex (hps, Options, RgbColor, *Index***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(BIT32)* — input
Options:

LCOLOPT_REALIZED
If this is specified, the information is required for when the logical color table is realized. (See GpiRealizeColorTable.)

If it is not specified (flag is not set) the information is required for when the logical color table (if any) is not realized.

Other bits are reserved, and must be zero.

RgbColor *(LONG)* — input
Specifies a color in RGB terms.

Index *(LONG)* — return
Color index providing closest match to the specified color:

≥0	Color index
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COLOR_OPTIONS
PMERR_INV_RGBCOLOR

Remarks

If an RGB logical color table has been loaded, this call returns the same RGB color that is passed to it.

GpiQueryCp – Query Code Page

This call returns the currently selected graphics code page.

GpiQueryCp (hps, *CodePage*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

CodePage *(USHORT)* – return
Code page:

GPI_ERROR Error
Otherwise Code page.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

The code page identity returned is the one that is set by GpiSetCp (or defaulted when the presentation space is first created). This is the code page of the default font, not the currently-selected font found from GpiQueryFontMetrics.

GpiQueryCurrentPosition — Query Current Position

This call returns the value of current position.

GpiQueryCurrentPosition (hps, Point, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — output
Current position.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryDefArcParams – Query Default Arc Parameters

This call returns the default values of the arc parameters, as set by the GpiSetDefArcParams call.

GpiQueryDefArcParams (hps, ArcParams, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

ArcParams *(ARCPARAM)* – output
Default arc parameters.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

GpiQueryDefAttrs — Query Default Attributes

This call returns default attribute values for the specified primitive type.

GpiQueryDefAttrs (hps, PrimType, AttrMask, Attrs, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

PrimType *(LONG)* — input
Primitive type.

This is the type of primitive for which default attribute values are to be queried, as follows:

PRIM_LINE	Line and arc primitives
PRIM_CHAR	Character primitives
PRIM_MARKER	Marker primitives
PRIM_AREA	Area primitives
PRIM_IMAGE	Image primitives.

AttrMask *(BIT32)* — input
Attributes mask.

Each flag that is set indicates that the default value of the corresponding attribute is to be returned in the **Attrs** buffer.

If all flags in **AttrMask** are zero, the **Attrs** buffer address is not used.

Attrs *(BUNDLE)* — output
Attributes.

Attrs is a buffer in which is returned the default value of each attribute for which the **AttrMask** flag is set, in the order specified in GpiSetAttrs for the particular primitive type.

Only data for attributes for which the appropriate flag in **AttrMask** is set is updated, so **Attrs** need only be large enough for the highest offset attribute to be returned (see GpiSetAttrs).

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PRIMITIVE_TYPE
PMERR_UNSUPPORTED_ATTR

Remarks

The parameters returned by this call may be used to reinstate exactly the same default attribute values as are queried, using GpiSetDefAttrs.

GpiQueryDefaultViewMatrix — Query Default View Matrix

SAA

This call returns the current default viewing transform; see GpiSetDefaultViewMatrix.

GpiQueryDefaultViewMatrix (hps, Count, Array, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of elements.

The number of elements to be returned in **Array**; must be in the range 0 through 9. If 0 is specified, no matrix elements are returned.

Array *(MATRIX)* — output
Transform matrix.

An array into which the elements of the default viewing transform matrix are returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT

GpiQueryDefCharBox — Query Default Graphics Character Box

This call returns the size of the default graphics character box in world coordinates.

GpiQueryDefCharBox (hps, Size, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Size *(SIZEROL)* — output
Default character-box size.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

GpiQueryDefTag – Query Default Tag

This call returns the default value of the tag identifier, as set by the GpiSetDefTag call.

GpiQueryDefTag (hps, Tag, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Tag *(LONG)* – output
Default tag identifier.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION

GpiQueryDefViewingLimits — Query Default Viewing Limits

This call returns the default value of the viewing limits, as set by the GpiSetDefViewingLimits call.

GpiQueryDefViewingLimits (hps, Limits, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Limits *(RECT)* — output
Default viewing limits.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

GpiQueryDevice — Query Device

This call returns the handle of the currently associated device context.

GpiQueryDevice (hps, *hdc*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

hdc *(HDC)* — return
Device-context handle:

HDC_ERROR	Error
NULL	No device context is currently associated
Otherwise	Device context handle.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_INV_HDC

GpiQueryDeviceBitmapFormats — Query Device Bit-Map Formats

This call returns the formats of bit maps supported internally by the device driver.

GpiQueryDeviceBitmapFormats (hps, Count, Array, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

The associated device context defines the class of device for which formats are required. This must be either a memory device context or a device context for a device that supports raster operations.

Count *(LONG)* — input
Number of elements.

Number of elements in **Array** (must be an even number). For the complete set of formats returned, the value of this parameter must be at least double the number of device formats returned by DevQueryCaps.

Array *(LONG****Count)*** — output
Data array.

Array of elements that, on return, is set to pairs of (**planes, bitcount**) (see *BITMAPINFOHEADER*), for each supported format in turn. Any unused elements are set to zero.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT

Remarks

An application can create, set, and query bit maps using any of the standard formats. Internally, however, these are converted by the device driver into one of the device internal formats if necessary. This is normally a smaller set than the standard set of bit-map formats.

The number of device bit-map formats can be found with DevQueryCaps (CAPS_BITMAP_FORMATS).

The first pair of (**planes, bitcount**) elements returned most closely matches the device.

This call must not be issued when there is no device context associated with the presentation space.

GpiQueryDrawControl — Query Draw Control

This call returns a drawing control as set by GpiSetDrawControl.

GpiQueryDrawControl (hps, Control, *Value***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Control *(LONG)* — input
Control whose value is to be returned:

DCTL_ERASE	Erase before draw
DCTL_DISPLAY	Display
DCTL_BOUNDARY	Accumulate boundary data
DCTL_DYNAMIC	Draw dynamic segments
DCTL_CORRELATE	Correlate.

Value *(LONG)* — return
Value of the control.

(See GpiSetDrawControl for details):

DCTL_OFF	Off
DCTL_ON	On
DCTL_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_DRAW_CONTROL
PMERR_INV_MICROPS_DRAW_CONTROL

GpiQueryDrawingMode — Query Drawing Mode

This call returns the current drawing mode, as set by GpiSetDrawingMode.

GpiQueryDrawingMode (hps, *Mode*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Mode *(LONG)* — return
Drawing mode.

(See GpiSetDrawingMode for details):

>0	Drawing mode
DM_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION

GpiQueryEditMode — Query Edit Mode

SAA

This call returns the current editing mode, as set by GpiSetEditMode.

```
GpiQueryEditMode (hps, Mode)
```

Parameters

hps *(HPS)* — input
Presentation-space handle.

Mode *(LONG)* — return
Current editing mode:

SEGEM_INSERT	Insert mode
SEGEM_REPLACE	Replace mode
SEGEM_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION

Remarks

This call can be issued in any drawing mode.

GpiQueryElement – Query Element

This call returns element content data.

GpiQueryElement (hps, Off, MaxLength, Data, *RetLength***)**

Parameters

hps *(HPS)* – input
Presentation-space handle.

Off *(LONG)* – input
Starting byte offset within the element.

MaxLength *(LONG)* – input
Maximum length of data that can be returned.

Data *(BUFFER)* – output
Element content data.

An area of **MaxLength** bytes in which the element content data is to be returned.

RetLength *(LONG)* – return
Number of bytes returned:

≥0	Actual number of bytes returned
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NO_CURRENT_ELEMENT
PMERR_NOT_IN_RETAIN_MODE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_IN_ELEMENT
PMERR_INV_ELEMENT_OFFSET
PMERR_NO_CURRENT_SEG

Remarks

Returns the element content (or part of the element content) for the element to which the element pointer currently points.

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress.

This call is not valid within an element bracket.

GpiQueryElementPointer — Query Element Pointer

SAA

This call returns the current element pointer.

GpiQueryElementPointer (hps, *Element*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Element *(LONG)* — return
Current element pointer:

≥0	Current element pointer
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NOT_IN_RETAIN_MODE
PMERR_NO_CURRENT_SEG

Remarks

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress.

GpiQueryElementType — Query Element Type

This call returns information about the element to which the element pointer currently points.

GpiQueryElementType (hps, Type, Length, Data, *ReqLength***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Type *(LONG)* — output
Element type.

The element type can be system-defined or application-defined; see GpiElement and GpiBeginElement.

Length *(LONG)* — input
Data length.

Length of the description data buffer.

Data *(STRL)* — output
Description of data buffer.

The description may be system-defined or application-defined; see GpiElement and GpiBeginElement. The string is null-terminated, even if it has to be truncated.

ReqLength *(LONG)* — return
Size of the data required to hold the element content.

This can be used for a subsequent GpiQueryElement call.

≥0	Size of data
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NO_CURRENT_ELEMENT
PMERR_NOT_IN_RETAIN_MODE
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_IN_ELEMENT
PMERR_NO_CURRENT_SEG

Remarks

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress. It is not valid in an element bracket.

GpiQueryFontFileDescriptions — Query Font File Descriptions

This call determines whether a given file is a font resource file, and if so returns the family and face names of the fonts that it contains.

GpiQueryFontFileDescriptions (hab, Filename, Count, Names, *RemFonts***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Filename *(STRL)* — input
Fully qualified filename.

This is the name of the font resource. The filename extension is ".FON."

Count *(LONG)* — input/output
Maximum number of family and facename pairs to be returned.

The number of pairs of descriptions that are actually returned in **Names** is returned in this variable.

Names *(FFDESCS****Count)** — output
Array of font file descriptors.

An array of 2***Count** consecutive 32-byte fields, in which the family and face names of each font, in turn, are returned alternately. For each pair, the family name is returned first.

RemFonts *(LONG)* — return
Returns:

≥0	Number of fonts for which details were not returned
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_FONT_FILE_DATA
PMERR_INV_LENGTH_OR_COUNT

Remarks

Details are returned for as many fonts as can be held in **Names**.

By inspecting the returned data, the application can tell whether a particular font resource file contains the fonts it requires, before loading it.

By specifying **Count** as **0**, and then looking at the value returned in **RemFonts**, an application can determine how many fonts there are in the file, and then allocate the correct amount of buffer space for a subsequent call to obtain all of the names.

GpiQueryFontMetrics — Query Font Metrics

This call returns a record providing details of the font metrics for the logical font that is currently selected.

GpiQueryFontMetrics (hps, MetricsLength, Metrics, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

MetricsLength *(LONG)* — input
Length of metrics.

Metrics *(FONTMETRICS)* — output
Metrics of font.

In this buffer are returned the font metrics of the logical font, identified by the current value of the character set attribute.

No more data than **MetricsLength** is returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_COORDINATE_OVERFLOW

Remarks

All sizes are returned in world coordinates.

GpiQueryFonts — Query Fonts

SAA

This call returns a record providing details of the fonts that match the specified **Facename**.

GpiQueryFonts (hps, Options, Facename, ReqFonts, MetricsLength, Metrics, *RemFonts***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(BIT32)* — input
Enumeration options.

This controls which fonts are to be enumerated. If both the following options are required, the values should be OR'ed together:

QF_PUBLIC Enumerate public fonts
QF_PRIVATE Enumerate private fonts.

Facename *(STRL)* — input
Facename of fonts.

If the pointer to **Facename** is NULL, all available fonts are queried, regardless of their face names.

ReqFonts *(LONG)* — input/output
Count of fonts.

The number of fonts for which the application requires the metrics. This variable returns the number of fonts returned.

MetricsLength *(LONG)* — input
Length of metrics.

The length of each metrics record to be returned. The **Metrics** data area must be **ReqFonts** multiplied by **MetricsLength** long.

Metrics *(FONTMETRICS****ReqFonts)*** — output
Metrics of font.

In this structure are returned the font metrics of up to **ReqFonts** matching fonts. The format for each record is as defined for GpiQueryFontMetrics, except that the **codepage** field has no meaning in this context, and is indeterminate. For each font, no more data than **MetricsLength** is returned.

RemFonts *(LONG)* — return
Count of fonts not returned:

≥0 Count of fonts not returned
GPI_ALTERROR Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_COORDINATE_OVERFLOW

Remarks

Font metrics are returned for as many matching fonts as can be held in **Metrics**.

By inspecting the returned data, the application can 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 **ReqFonts** as **0**, and then looking at the value returned in **RemFonts**, an application can determine how many fonts there are that match the **Facename**.

All sizes are returned in world coordinates.

If the device has fonts that cannot be positioned to the nearest pel, these are not normally reported by GpiQueryFonts. However, if the DEVESC_DRAFTMODE escape has been issued (see DevEscape), any such fonts will be returned. No special indicator is returned to show that the fonts cannot be accurately positioned.

GpiQueryGraphicsField — Query Graphics Field

SAA

This call returns the bottom-left and top-right corners of the graphics field in presentation page units, as set by the GpiSetGraphicsField call.

GpiQueryGraphicsField (hps, Field, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Field *(RECT)* — output
Graphics field.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

GpiQueryInitialSegmentAttrs — Query Initial Segment Attributes

This call returns the current value of a particular initial segment attribute.

GpiQueryInitialSegmentAttrs (hps, Attribute, *Value*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Attribute *(LONG)* — input
Attribute to be queried.

Identifies the attribute to be returned by this call:

ATTR_DETECTABLE	Detectability
ATTR_VISIBLE	Visibility
ATTR_CHAINED	Chained
ATTR_DYNAMIC	Dynamic
ATTR_FASTCHAIN	Fast chaining
ATTR_PROP_DETECTABLE	Propagate detectability
ATTR_PROP_VISIBLE	Propagate visibility.

Value *(LONG)* — return
Current initial attribute value:

ATTR_ON	On/yes
ATTR_OFF	Off/no
ATTR_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_ATTR
PMERR_INV_MICROPS_FUNCTION

Remarks

Initial segment attributes are modal settings used to determine the initial attributes of new segments as those new segments are created; see GpiSetInitialSegmentAttrs.

GpiQueryKerningPairs — Query Kerning Pairs

SAA

This call returns kerning pair information for the logical font identified by the current value of the character set attribute.

GpiQueryKerningPairs (hps, Count, Data, *Returned*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
The number of elements in **Data**.

Data *(KERNINGPAIRS*Count)* — output
Kerning pairs.

An array of **Count** kerning pairs in which information is returned. No more than **Count** records are returned.

Returned *(LONG)* — return
Number returned/error indicator:

≥0	Number of kerning pairs returned
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_COORDINATE_OVERFLOW

Remarks

The number of kerned pairs is a field in the font metrics.

GpiQueryLineEnd — Query Line End

This call returns the current line-end attribute.

GpiQueryLineEnd (hps, *LineEnd*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineEnd *(LONG)* — return
Line end:

LINEEND_DEFAULT	Default
>0	Line end
LINEEND_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryLineJoin — Query Line Join

SAA

This call returns the current line-join attribute, as set by the GpiSetLineJoin call.

GpiQueryLineJoin (hps, *LineJoin*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineJoin *(LONG)* — return
Line join:

LINEJOIN_DEFAULT	Default
>0	Line join
LINEJOIN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryLineType — Query Line Type

This call returns the current cosmetic line-type attribute, as set by the GpiSetLineType call.

GpiQueryLineType (hps, *LineType***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineType *(LONG)* — return
Line type:

LINETYPE_DEFAULT	Default
>0	Line type
LINETYPE_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryLineWidth — Query Line Width

SAA

This call returns the current value of the cosmetic line-width attribute, as set by the GpiSetLineWidth call.

GpiQueryLineWidth (hps, *LineWidth***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineWidth *(ROF)* — return
Line width:

LINEWIDTH_DEFAULT	Default
>0	Line width
LINEWIDTH_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryLineWidthGeom — Query Line Width Geom

This call returns the current geometric line-width attribute, as set by the GpiSetLineWidthGeom call.

GpiQueryLineWidthGeom (hps, *LineWidth*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineWidth *(ROL)* — return
Geometric line width:

≥0	Geometric line width
LINEWIDTHGEOM_ERROR	Error.

If the geometric line width is currently set to the default, zero is returned.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryLogColorTable – Query Logical Color Table

This call returns the logical color table.

GpiQueryLogColorTable (hps, Options, Start, Count, Array, *RetCount***)**

Parameters

hps *(HPS)* – input
Presentation-space handle.

Options *(BIT32)* – input
Specifies options:

LCOLOPT_INDEX

B'1'	The index is to be returned for each RGB value
Other	Reserved; must be B'0'.

Start *(LONG)* – input
The starting index for which data is to be returned.

Count *(LONG)* – input
Count of elements.

Number of elements available in **Array**.

Array *(LONG****Count)** – output
An array in which the information is returned.

If the LCOLOPT_INDEX flag is B'0', it is an array of RGB 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 **Array** has been exhausted. If the logical color table is not loaded with a contiguous set of indices, CLR_NOINDEX is returned as the RGB value for any index values, outside the default range, that have not been explicitly loaded.

If the LCOLOPT_INDEX flag is B'1', it is an array of alternating color indices and RGB values, in the order index1, RGB value1, index2, RGB value2,... An even number of elements is always returned. If the logical color table is not loaded with a contiguous set of indices, any index values that are not loaded are skipped.

RetCount *(LONG)* – return
Number of elements returned/error indicator:

QLCT_RGB	Table in RGB mode, no elements returned
>0	Number of elements returned
QLCT_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX

GpiQueryMarker — Query Marker

This call returns the current value of the marker symbol attribute, as set by the GpiSetMarker call.

GpiQueryMarker (hps, *Symbol***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Symbol *(LONG)* — return
Marker symbol:

MARKSYM_DEFAULT	Default
>0	Marker symbol
MARKSYM_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryMarkerBox — Query Marker Box

SAA

This call returns the current value of the marker-box attribute, as set by the GpiSetMarkerBox call.

GpiQueryMarkerBox (hps, Size, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Size *(SIZEOF)* — output
Size of marker box.

The size of the marker box is in world coordinates.

If the marker box is currently set to the default, the default size is returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryMarkerSet — Query Marker Set

This call returns the current value of the marker-set attribute, as set by the GpiSetMarkerSet call.

GpiQueryMarkerSet (hps, *Set*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Set *(LONG)* — return
Marker-set local identifier:

LCID_DEFAULT	Default
>0	Marker-set local identifier
LCID_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryMetaFileBits — Query Metafile Bits

SAA

This call transfers a metafile to application storage.

GpiQueryMetaFileBits (hmf, Offset, Length, Data, *Success***)**

Parameters

hmf *(HMF)* — input
Memory-metafile handle.

Offset *(LONG)* — input
Byte offset.

Offset into the metafile data from which the transfer must start. This is useful in instances where the metafile data is too long to fit into a single application buffer.

Length *(LONG)* — input
Length in bytes of the metafile data to copy.

Data *(BUFFER)* — output
Metafile data.

Address in application storage into which the metafile data is copied.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HMF
PMERR_INV_METAFILE_LENGTH
PMERR_INV_METAFILE_OFFSET
PMERR_METAFILE_IN_USE
PMERR_TOO_MANY_METAFILES_IN_USE

Remarks

The total length of a metafile can be found from the data returned by GpiQueryMetaFileLength. This call allows an application to retrieve the data in units of a manageable size.

GpiQueryMetaFileLength — Query Metafile Length

This call returns the total length of a memory metafile, in bytes.

GpiQueryMetaFileLength (hmf, *Length*)

Parameters

hmf *(HMF)* — input
Memory-metafile handle.

Length *(LONG)* — return
Total length of the metafile:

≥0	Total length of the metafile
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HMF
PMERR_TOO_MANY_METAFILES_IN_USE

Remarks

This call is normally used before GpiQueryMetaFileBits.

GpiQueryMix — Query Mix

SAA

This call returns the current value of the (character) foreground color-mixing mode, as set by the GpiSetMix call.

GpiQueryMix (hps, *MixMode*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

MixMode *(LONG)* — return
Mix mode:

FM_DEFAULT	Default
>0	Mix mode
FM_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryModelTransformMatrix — Query Model Transform Matrix

This call returns the current model transform; see GpiSetModelTransformMatrix.

GpiQueryModelTransformMatrix (hps, Count, Array, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of elements.

The number of elements to be returned in **Array**; must be in the range 0 through 9. If 0 is specified, no matrix elements are returned.

Array *(MATRIX)* — output
Transform matrix.

A structure in which the elements of the model transform matrix are returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE
PMERR_INV_LENGTH_OR_COUNT

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryNearestColor — Query Nearest Color

SAA

This call returns the nearest color available to the color specified on the currently associated device. Both colors are specified in RGB terms.

GpiQueryNearestColor (hps, Options, RgbIn, *RgbOut*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(BIT32)* — input
Options:

LCOLOPT_REALIZED
If this is specified, the information is required for when the logical color table is realized. (See GpiRealizeColorTable.)

If it is not specified (flag is not set) the information is required for when the logical color table (if any) is not realized.

Other bits are reserved, and must be zero.

RgbIn *(LONG)* — input
Required color.

RgbOut *(LONG)* — return
Nearest available color to the one specified:

≥0	Nearest available color
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COLOR_OPTIONS
PMERR_INV_RGBCOLOR

Remarks

The nearest color returned is one that is available in the *physical* palette on the device. This might not actually be available with the currently loaded logical color table.

The color returned is a pure color, that is, one that can be used for drawing lines, text, and so on. It does not take into account the possibility of *dithered* colors being used for filled areas. With *dithering*, it is likely that the color used for filling areas is different from that used for lines, and text, when the same color index is selected.

For a monochrome device, if **RgbIn** is the reset color, then **RgbOut** is also the reset color; otherwise, it is black if the reset color is white, and the converse.

GpiQueryNumberSetIds — Query Number Set Identifiers

This call returns the number of local identifiers (lcids) currently in use, referring to fonts or bit maps.

GpiQueryNumberSetIds (hps, *Count*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — return
Number of lcids:

≥0	Number of lcids in use
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

The information returned by this call can be used to perform a subsequent GpiQuerySetIds request.

GpiQueryPageViewport — Query Page Viewport

SAA

This call returns the page viewport; see GpiSetPageViewport.

GpiQueryPageViewport (hps, Viewport, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Viewport *(RECT)* — output
Page viewport.

The size and position of the page viewport in device units.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call must not be issued when there is no device context associated with the presentation space.

GpiQueryPattern — Query Pattern

This call returns the current value of the shading-pattern symbol, as set by the GpiSetPattern call.

GpiQueryPattern (hps, *PatternSymbol***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

PatternSymbol *(LONG)* — return
Pattern symbol:

PATSYM_DEFAULT	Default
>0	Pattern symbol
PATSYM_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryPatternRefPoint — Query Pattern Reference Point

SAA

This call returns the current pattern reference point, as set by the GpiSetPatternRefPoint call.

GpiQueryPatternRefPoint (hps, RefPoint, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

RefPoint *(POINT)* — output
Pattern reference point.

If the pattern reference point is currently set to the default, (0,0) is returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryPatternSet — Query Pattern Set

This call returns the current value of the pattern-set identifier, as set by the GpiSetPatternSet call.

GpiQueryPatternSet (hps, *Set*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Set *(LONG)* — return
Pattern-set local identifier:

LCID_DEFAULT	Default
>0	Pattern set
LCID_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is not valid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryPel – Query Pel

SAA

This call returns the color of a pel at a position specified in world coordinates.

GpiQueryPel (hps, Point, *Color*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Point *(POINT)* – input
Position in world coordinates.

It is an error if the specified point is outside any of the current clipping objects (that is, clip path, viewing limits, clip region, or visible region).

Color *(LONG)* – return
Color index of the pel:

≥0	Color of the pel
CLR_NOINDEX	No valid index (color is not in logical color table)
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_PEL_IS_CLIPPED
PMERR_PEL_NOT_AVAILABLE
PMERR_NO_BITMAP_SELECTED

Remarks

The color returned is a color index or RGB value, according to the logical color table in force (see GpiCreateLogColorTable).

GpiQueryPickAperturePosition — Query Pick Aperture Position

This call returns the position of the center of the pick aperture.

GpiQueryPickAperturePosition (hps, Point, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — output
Pick-aperture position.

Position of the center of the pick aperture, in presentation-page coordinates.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

GpiQueryPickApertureSize — Query Pick Aperture Size

SAA

This call returns the value of the pick-aperture size, as set by the GpiSetPickApertureSize call.

GpiQueryPickApertureSize (hps, Size, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Size *(SIZEROL)* — output
Pick-aperture size.

Size of the pick aperture, in presentation-page coordinates.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

GpiQueryPS — Query Presentation Space

This call returns page parameters for the presentation space.

GpiQueryPS (hps, Size, *Options***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Size *(SIZEROL)* — output
Presentation-page size.

Options *(BIT32)* — return
Presentation-space options.

For details, see the GpiCreatePS call.

The individual fields of the presentation-space options can be extracted by ANDing the returned value with the appropriate constant.

The **PS_ASSOCIATE** field of **Options** (see GpiCreatePS) should not be used on this call. The value of this field is not necessarily the same value that is specified when the presentation space is created.

PS_UNITS	Presentation-space size units
PS_FORMAT	Presentation-space coordinate format
PS_TYPE	Presentation-space type
PS_MODE	Presentation-space mode.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

GpiQueryRealColors — Query Real Colors

SAA

This call returns the RGB values of the distinct colors available on the currently associated device.

GpiQueryRealColors (hps, Options, Start, Count, Colors, *RetCount*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(BIT32)* — input
Options:

Either or both of these values can be specified (if both are specified, the values must be ORed together). If neither are required, the parameter must be zero.

LCOLOPT_REALIZED	If this is specified, the information is required for when the logical color table is realized. (See GpiRealizeColorTable.) If it is not specified (flag is not set) the information is required for when the logical color table (if any) is not realized.
LCOLOPT_INDEX	If this is specified, the index is to be returned for each RGB value. If this flag is set when RGB mode is in force (LCOLF_RGB is set on GpiCreateLogColorTable), the RGB value is returned as the index. Any color not available with the current logical color table is given a special index value of CLR_NOINDEX. If it is not specified (flag is not set) index values are not returned.
Other	Other bits are reserved, and must be zero.

Start *(LONG)* — input
Ordinal number of the first color required.

To start the sequence, this parameter is set to zero.

Programming Note: This parameter is not the color index, and the order in which the colors are returned is not defined.

Count *(LONG)* — input
Maximum number of elements.

Number of elements available in **Colors**.

Colors *(LONG****Count)*** — output
Array in which the information is returned.

Contents depend on the setting of the LCOLOPT_INDEX flag:

0 An array of color values (each value is as defined for GpiCreateLogColorTable).
1 An array of alternating color indexes and values, in the order index1, value1, index2, value2,... indexn, valuen. An even number of elements is always returned in this case.

RetCount *(LONG)* — return
Number of elements returned:

≥0	Number of elements returned
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_START_INDEX

Remarks

Subject to space in the **Colors** parameter, all colors that are physically available on the device are returned.

GpiQueryRegionBox — Query Region Box

SAA

This call returns the dimensions of the smallest rectangle able to bound the region.

GpiQueryRegionBox (hps, hrgn, Bound, *Complexity***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

The region must be owned by the device identified by the currently associated device context.

hrgn *(HRGN)* — input
Region handle.

Bound *(RECT)* — output
Bounding rectangle.

Complexity *(LONG)* — return
Complexity of region/error indicator:

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_REGION_IS_CLIP_REGION
PMERR_HRGN_BUSY

Remarks

If the region is null, the rectangle returned has the left boundary equal to the right, and the top boundary equal to the bottom.

It is invalid if the specified region is currently selected as the clip region (by GpiSetClipRegion).

GpiQueryRegionRects — Query Region Rectangles

This call returns the rectangles that, when OR'ed together, define the specified region.

GpiQueryRegionRects (hps, hrgn, Bound, Control, Rects, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

The region must be owned by the device identified by the currently associated device context.

hrgn *(HRGN)* — input
Region handle.

Bound *(RECT)* — input
Bounding rectangle.

NULL Return all the rectangles in the region.
Other Return only rectangles that intersect with the bounding rectangle. Each rectangle returned is the intersection of the bounding rectangle with a rectangle in the region.

Control *(RGNRECT)* — input/output
Processing-control structure.

Rects *(RECT*count)* — output
Array of rectangle structures, in which the rectangles are returned.

The maximum number of rectangles that can be returned is specified by the **count** parameter of the *RGNRECT* structure identified by the **Control** parameter.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_INV_REGION_CONTROL
PMERR_INV_COORDINATE
PMERR_INV_RECT
PMERR_REGION_IS_CLIP_REGION
PMERR_HRGN_BUSY

Remarks

Points on the right-hand and top boundaries are not included in the region. Points on the left-hand and bottom boundaries, that are not also on the right-hand or top boundaries (that is, the top-left and bottom-right corner points), are included.

It is invalid if the specified region is currently selected as the clip region (by GpiSetClipRegion).

GpiQueryRGBColor — Query RGB Color

SAA

This call returns the actual RGB color that results from a particular index on the currently-associated device.

GpiQueryRGBColor (hps, Options, ColorIndex, *RgbColor*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(BIT32)* — input
Options:

LCOLOPT_REALIZED
If this is specified, the information is required for when the logical color table is realized. (See GpiRealizeColorTable.)

If it is not specified (flag is not set) the information is required for when the logical color table (if any) is not realized.

Other bits are reserved, and must be zero.

ColorIndex *(LONG)* — input
Color index.

This can be any normally valid color index value (see GpiSetColor) except CLR_DEFAULT.

RgbColor *(LONG)* — return
RGB color providing closest match to the specified color index:

≥0	RGB color providing closest match
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COLOR_OPTIONS
PMERR_INV_COLOR_INDEX

Remarks

If an RGB logical color table has been loaded, this call returns the nearest RGB color. This call is then equivalent to GpiQueryNearestColor.

GpiQuerySegmentAttrs — Query Segment Attributes

This call returns the current value of the specified attribute as set by the GpiSetSegmentAttrs call.

GpiQuerySegmentAttrs (hps, Segid, Attribute, *Value***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Segid *(LONG)* — input
Segment identifier; must be greater than 0.

The name of the segment for which attribute information is to be returned.

Attribute *(LONG)* — input
Attribute to be queried.

For details of the following attributes, see the GpiSetInitialSegmentAttrs call.

Identifies the attribute of the segment to be returned by this function:

ATTR_DETECTABLE	Detectability
ATTR_VISIBLE	Visibility
ATTR_CHAINED	Chained
ATTR_DYNAMIC	Dynamic
ATTR_FASTCHAIN	Fast chaining
ATTR_PROP_DETECTABLE	Propagate detectability
ATTR_PROP_VISIBLE	Propagate visibility.

Value *(LONG)* — return
Current attribute value:

ATTR_ON	On/yes
ATTR_OFF	Off/no
ATTR_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_SEG_ATTR
PMERR_SEG_NOT_FOUND
PMERR_INV_MICROPS_FUNCTION

Remarks

The segment can be any retained segment, including the currently open one if this is retained.

GpiQuerySegmentNames — Query Segment Names

SAA

This call returns the identifiers of all segments that exist with identifiers in a specified range.

GpiQuerySegmentNames (hps, FirstSegId, LastSegId, Max, SegIds, *RetCount*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

FirstSegId *(LONG)* — input
First segment in the range (must be greater than 0).

LastSegId *(LONG)* — input
Last segment in the range (must be greater than 0).

Max *(LONG)* — input
Maximum number.

This is the maximum number of segment identifiers to be returned in **SegIds**.

SegIds *(LONG*Max)* — output
Array in which the required identifiers are returned.

RetCount *(LONG)* — return
Number of identifiers returned:

≥0	Number of identifiers returned
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MICROPS_FUNCTION

Remarks

Nonretained segment identifiers are not returned. If **FirstSegId** is the same as, or greater than **LastSegId**, the search terminates after querying only the segment with **FirstSegId**.

GpiQuerySegmentPriority — Query Segment Priority

This call returns the identifier of the named segment that is chained immediately before or after a specified reference segment.

GpiQuerySegmentPriority (hps, RefSegId, Order, *Segid***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

RefSegId *(LONG)* — input
Reference-segment identifier.

Order *(LONG)* — input
Segment higher or lower.

Shows whether a segment identifier of a higher or lower priority than that identified in the **RefSegId** parameter is to be returned. Possible values are:

LOWER_PRI Return the next segment with a lower priority than **RefSegId**. If **RefSegId** = 0, query the identifier of the segment with the lowest priority.
HIGHER_PRI Return the next segment with a higher priority than **RefSegId**. If **RefSegId** = 0, query the identifier of the segment with the highest priority.

Segid *(LONG)* — return
Segment identifier.

The identifier of the segment that is immediately before or after that in the **RefSegId** parameter:

>0 Segment identifier.
0 The segment specified in the **RefSegId** parameter is either the lowest-priority segment (when **Order** = LOWER_PRI) or the highest-priority segment (when **Order** = HIGHER_PRI).
GPI_ALTERROR Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_ORDERING_PARM
PMERR_SEG_NOT_CHAINED
PMERR_SEG_NOT_FOUND
PMERR_INV_MICROPS_FUNCTION

Remarks

The segment that is chained before the specified segment, is considered to have a lower priority than the specified segment; similarly, the segment that is chained after the specified segment, is considered to have a higher priority than the specified segment.

Unnamed segments (with an identifier of zero) are ignored.

GpiQuerySegmentTransformMatrix — Query Segment Transform Matrix

SAA

This call returns the elements of the transform of the identified segment (see GpiSetSegmentTransformMatrix).

GpiQuerySegmentTransformMatrix (hps, SegId, Count, Array, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

SegId *(LONG)* — input
Segment identifier.

Count *(LONG)* — input
Number of elements.

The number of elements that are to be set in the **Array** parameter. **Count** must be in the range 0 through 9.

Array *(MATRIX)* — output
Transform matrix.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_LENGTH_OR_COUNT
PMERR_SEG_NOT_FOUND

GpiQuerySetIds — Query Set Identifiers

This call returns information about all the fonts that have been created by GpiCreateLogFont, and tagged bit maps (see GpiSetBitmapId).

GpiQuerySetIds (hps, Count, Types, Names, lcids, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
The number of objects to be queried.

The number of local identifiers (lcids) currently in use, and therefore the maximum number of objects for which information can be returned, can be found with GpiQueryNumberSetIds.

Types *(LONG*Count)* — output
Object types.

Elements indicate whether the corresponding **lcids** element refers to a logical font or a tagged bit map.

LCIDT_FONT Font object
LCIDT_BITMAP Bit map.

Names *(STR8*Count)* — output
Font names.

An array of **Count** consecutive 8-byte fields, in which the 8-character names associated with the logical fonts are returned. For bit maps, the whole field is set to zeros.

lcids *(LONG*Count)* — output
Local identifiers.

An array in which the local identifier (lcid) values are returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT

Remarks

Each of the output parameters is an array with **Count** elements. Information about the first **Count** objects is returned; if there are fewer than **Count**, the **Types** and **lcids** elements for the remainder are cleared to zero.

GpiQueryStopDraw — Query Stop Draw

This call indicates whether the "stop draw" condition currently exists.

GpiQueryStopDraw (hps, *Value*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Value *(LONG)* — return
Stop draw condition indicator:

SDW_OFF	No "stop draw" condition currently exists
SDW_ON	The "stop draw" condition currently exists
SDW_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_INV_MICROPS_FUNCTION

Remarks

See GpiSetStopDraw for details.

GpiQueryTag – Query Tag

This call returns the current value of the tag identifier, as set by the GpiSetTag call.

GpiQueryTag (hps, Tag, *Success***)**

Parameters

hps *(HPS)* – input
Presentation-space handle.

Tag *(LONG)* – output
Tag identifier.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain.**

GpiQueryTextBox — Query Text Box

SAA

This call returns the relative coordinates of the four corners of a text box.

GpiQueryTextBox (hps, Count1, String, Count2, Points, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count1 *(LONG)* — input
Number of characters.

String *(STR)* — input
The character string.

Count2 *(LONG)* — input
Number of points.

Contains the number of points to be returned in the **Points** array. Specify TXTBOX_COUNT to get the maximum information.

Points *(POINT*Count2)* — output
List of points.

The list of points contains the relative coordinates of the text box in world coordinates. The array elements are numbered consecutively, starting with TXTBOX_TOPLEFT. The element number constants start either with 0 or 1 depending on the language. Refer to the appropriate bindings reference. A **Count2** value of TXTBOX_COUNT will cause all of the defined array elements to be returned.

The terms 'top-left', 'bottom-right', and so on, are well defined when the character angle is such that the baseline is parallel to the x axis and running left to right, and there is no character shear. If the character string is rotated or sheared, the term top-left applies to the corner of the box that appears in the top-left position when no rotation or shear is applied.

This is an example:

```
Set character angle = -1,1
String = ABCDE
```

Coordinates returned are as shown:

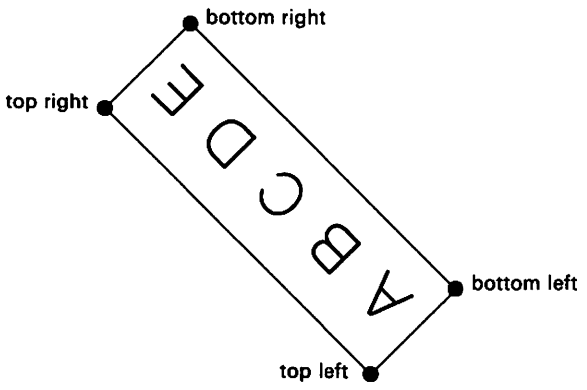


Figure 4-4. Box Enclosing Characters

TXTBOX_TOPLEFT	Top-left corner
TXTBOX_BOTTOMLEFT	Bottom-left corner
TXTBOX_TOPRIGHT	Top-right corner
TXTBOX_BOTTOMRIGHT	Bottom-right corner
TXTBOX_CONCAT	Concatenation point.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE
PMERR_INV_LENGTH_OR_COUNT
PMERR_COORDINATE_OVERFLOW

Remarks

The text box is defined as the parallelogram that encloses the specified character string when displayed on the device. Also returned are the relative coordinates of the concatenation point; that is, the value of current position after an equivalent GpiCharStringAt call. All coordinates are relative to the start point (see GpiSetCharDirection call). These coordinates can be used to box or underline the string, or to change the attributes in the middle of a longer string.

Programming Note: The height of the string is based on the maximum height of the font (including space for descenders, accents, and so on), not the maximum height of the actual characters in the string.

Character attributes are taken into account as if the string is to be drawn, but no output actually occurs. However, if the character mode (see GpiSetCharMode) is CM_MODE2 this function should only be used if the character angle (see GpiSetCharAngle), character direction (see GpiSetCharDirection) and character shear (see GpiSetCharShear) attributes are set to their default values.

This call is not valid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryViewingLimits — Query Viewing Limits

SAA

This call returns the current value of the viewing limits, as set by the GpiSetViewingLimits call.

```
GpiQueryViewingLimits (hps, Limits, Success)
```

Parameters

hps *(HPS)* — input
Presentation-space handle.

Limits *(RECT)* — output
Viewing limits.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_IN_RETAIN_MODE

Remarks

This call is invalid when the drawing mode (see GpiSetDrawingMode) is set to **retain**.

GpiQueryViewingTransformMatrix — Query Viewing Transform Matrix

This call returns the current viewing transform (see GpiSetViewingTransformMatrix).

GpiQueryViewingTransformMatrix (hps, Count, Array, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of elements.

The number of elements to be returned in **Array** (must be in the range 0 through 9). If 0 is specified, no matrix elements are returned.

Array *(MATRIX)* — output
Transform matrix.

A structure in which the elements of the viewing transform matrix are returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_LENGTH_OR_COUNT

GpiQueryWidthTable — Query Font Width Table

SAA

This call returns width table information for the logical font identified by the value of the character-set attribute.

GpiQueryWidthTable (hps, FirstChar, Count, Data, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

FirstChar *(LONG)* — input
Code point of first character.

The code point of the initial character, for which width-table information is required.

Count *(LONG)* — input
Count of elements in **Data**.

The number that should be allowed for, so as to retrieve the full width table. Data for this font can be found by GpiQueryFontMetrics.

Data *(ROL****Count)** — output
Array of width values.

An array of **Count** elements, in which width-table information is returned. No more than **Count** elements are returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_FIRST_CHAR
PMERR_INV_LENGTH_OR_COUNT
PMERR_COORDINATE_OVERFLOW

GpiRealizeColorTable — Realize Color Table

This call realizes the logical color table.

GpiRealizeColorTable (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_COL_TABLE_NOT_REALIZABLE

Remarks

This call causes the system to ensure, where possible, 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, this call should only be issued when the associated window is maximized.

An error is returned if an attempt is made to realize the color table on a device that does not support the call (this can be found from the CAPS_COLOR_TABLE_SUPPORT field in DevQueryCaps).

If the device (for example, a printer) implicitly realizes the color table when it is created, this call completes without error.

GpiRectInRegion — Rectangle In Region

SAA

This call checks whether any part of a rectangle lies within the specified region.

GpiRectInRegion (hps, hrgn, Rect, *Inside*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The region must be owned by the device identified by the currently associated device context.

hrgn *(HRGN)* — input
Region handle.

Rect *(RECT)* — input
Test rectangle.

The rectangle is specified in device coordinates.

Inside *(LONG)* — return
Inside/error indicator:

RRGN_OUTSIDE	Not in region
RRGN_PARTIAL	Some in region
RRGN_INSIDE	All in region
RRGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_INV_COORDINATE
PMERR_INV_RECT
PMERR_REGION_IS_CLIP_REGION
PMERR_HRGN_BUSY

Remarks

It is invalid if the specified region is currently selected as the clip region (by GpiSetClipRegion).

GpiRectVisible — Rectangle Visible

This call checks whether any part of a rectangle lies within the clipping region of the device associated with the specified presentation space.

GpiRectVisible (hps, Rectangle, *Visibility*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Rectangle *(RECT)* — input
Test rectangle, in world coordinates.

Points on the borders of the rectangle are considered to be included within the rectangle.

Visibility *(LONG)* — return
Visibility indicator:

RVIS_INVISIBLE	Not visible
RVIS_PARTIAL	Some of the rectangle is visible
RVIS_VISIBLE	All of the rectangle is visible
RVIS_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE
PMERR_INV_RECT

Remarks

For the purposes of this call, the clipping region is defined as the intersection between the application clipping region and any other clipping, including windowing.

GpiRemoveDynamics — Remove Dynamics

This call removes those parts of the displayed image that are drawn from the dynamic segments in a section of the segment chain. This includes any parts that are drawn by calls from these dynamic segments.

GpiRemoveDynamics (hps, FirstSegId, LastSegId, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

FirstSegId *(LONG)* — input
First segment in the section.

It must be greater than 0.

LastSegId *(LONG)* — input
Last segment in the section.

It must be greater than 0.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
PMERR_INV_FOR_THIS_DC_TYPE
DPC errors

Remarks

This call usually indicates that a dynamic segment is about to be updated; and that, having completed the update, GpiDrawDynamics is called to redraw the dynamic segments.

If there is more than one dynamic segment, only those that are being updated need be removed. The section of the segment chain is identified by the first and last segments in the section. If **FirstSegId** and **LastSegId** have the same value, this call erases only the parts drawn from the segment, and by calls from that segment.

Specifying the range of segment identifiers that are to be removed usually has a performance advantage, in that searching of the chain stops after **LastSegId** has been processed. It can also be used to operate on less than the maximum number of dynamic segments, as in one of the following examples:

- several dynamic segments are currently drawn, but only one is to be updated. Identifying this segment with both **FirstSegId** and **LastSegId** means that only this one is removed. It can then be updated, and replaced with GpiDrawDynamics.
- a new dynamic segment can be created, while the rest remain drawn. GpiRemoveDynamics is issued before the segment has been created (or while it is unchained, if it already exists), identifying it with both **FirstSegId** and **LastSegId**. It is then created with this identifier (or chained, if it already exists), and GpiDrawDynamics is issued, causing it to be drawn.

In these examples, the other dynamic segments remain drawn throughout.

In all cases, after GpiDrawChain, GpiDrawDynamics, GpiDrawFrom, or GpiDrawSegment where the DCTL_DYNAMIC draw control is set (see GpiSetDrawControl), **all** dynamic segments must be drawn. The **FirstSegId** and **LastSegId** parameters of GpiRemoveDynamics, cannot be used to cause a subset of dynamic segments to be drawn after the following GpiDrawDynamics. If this is required, it can be accomplished by unchaining the unwanted dynamic segments after first removing them.

Dynamic segments that are currently drawn must never be updated in the segment store, nor must any drawing in mix modes (other than exclusive-OR or leave-alone) be carried out to a presentation space while dynamic segments are drawn in it.

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

It is necessary to ensure that attributes, model transform, current position, and viewing limits are reset to their default values, before processing the segments. This can either be accomplished by ensuring that the first dynamic segment in the removed section does not have the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), or by issuing GpiResetPS before the GpiRemoveDynamics. The latter method also resets the clip path to cause no clipping, which may also be necessary.

If this call is followed by primitives or attributes, without first opening a segment, the processing is as described for GpiCloseSegment. In particular, note that during GpiRemoveDynamics, the system forces the foreground mix to FM_XOR and the background mix to BM_LEAVEALONE. It may be necessary to set one or both of these before proceeding to draw.

If **FirstSegId** does not exist, or is not in the segment chain, no removal or drawing occurs. However, the segment identifier range is still established for a subsequent GpiDrawDynamics call.

If **LastSegId** does not exist, or is not in the chain, or is chained before **FirstSegId**, no error is raised, and processing continues to the end of the chain.

GpiResetBoundaryData — Reset Boundary Data

This call resets the boundary data to null.

```
GpiResetBoundaryData (hps, Success)
```

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

This call is only necessary for **draw** mode (see GpiSetDrawingMode) boundary determination. Boundary data is automatically reset before any retained drawing call.

After drawing, boundary data can be found by issuing GpiQueryBoundaryData.

Note: Boundary data is not reset at the start of a segment.

GpiResetPS — Reset Presentation Space

This call resets the presentation space.

GpiResetPS (hps, Options, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(BIT32)* — input
Reset option:

GRES_ATTRS
This has the following effects:
- All current attributes and arc parameters are reset to their default values
- The current tag is reset to its default value
- The current model transform is reset to unity
- The current position is set to (0,0)
- Any open path or area is aborted
- Any open element bracket is closed
- Any open segment is closed
- The current clip path is set so as to cause no clipping
- The current viewing limits are reset to their default values.

GRES_SEGMENTS
This has all the effects of GRES_ATTRS plus:
- Any retained segments are deleted
- Initial segment attributes are reset to their default values
- The default viewing transform and the graphics field are reset to their default values
- The viewing transform is set to unity
- Drawing mode, draw controls, edit mode and attribute mode are reset to default values
- Boundary data is reset
- The currently selected clip region, if any, is deselected, and destroyed
- The default values of primitive attributes, arc parameters, viewing limits and primitive tag are reset to their initial values.

GRES_ALL
This has all the effects of GRES_ATTRS and GRES_SEGMENTS plus:
- Any logical fonts and local identifiers for bit maps are deleted
- Any loaded logical color table is reset to default.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_RESET_OPTIONS

GpiResetPS — Reset Presentation Space

Remarks

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

- As if a new (root) segment is being processed
- As if the presentation space is being created without deleting resources
- As if the presentation space is being created with resources deleted.

More details are provided under the description of **Options** above.

None of these options cause any drawing or erasing to take place on the device (GpiErase can be used to do the latter), nor is any association between the specified presentation space and a device context affected. The page viewport is also unaffected.

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiRestorePS — Restore Presentation Space

This call restores the state of the presentation space to the one that exists when the corresponding GpiSavePS is issued.

GpiRestorePS (hps, PSId, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

PSId *(LONG)* — input
Identifier of the saved presentation space that is to be restored:

- **>0** **PSId** must be the identifier of a saved presentation space on the stack. It is an error if it does not exist.
- **0** Is an error. (This might have resulted from an invalid use of GpiSavePS.)
- **<0** The absolute value of **PSId** indicates how many saved presentation spaces on the stack are required. Thus **−1** means that the most recently saved one is to be restored. It is an error if the absolute value is larger than the number of entries on the stack.

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

Success *(BOOL)* — return
Success indicator:

- **TRUE** Successful completion
- **FALSE** Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_NOT_IN_DRAW_MODE
PMERR_INV_ID

Remarks

The most recently saved presentation space need not be the one that is restored. In this case, any that are skipped over on the stack are discarded.

Any clip regions selected into discarded presentation spaces are automatically destroyed.

This call is valid in an open element bracket, and in an open segment bracket if the drawing mode (see GpiSetDrawingMode) is set to **draw**. If it occurs within an open area or path bracket, the corresponding GpiSavePS must have taken place earlier in the same bracket.

GpiRotate – Rotate Transform

This call applies a rotation to a transform matrix.

GpiRotate (hps, Array, Options, Angle, Center, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Array *(MATRIX)* – input/output
Transform matrix.

The elements of the transform, in row order. The first, second, fourth, and fifth elements are of type FIXED, and have an assumed binary point between the second and third bytes. Thus a value of 1.0 is represented by 65 536. Other elements are normal signed integers.

The third, sixth, and ninth elements must be 0, 0, and 1, respectively.

Options *(LONG)* – input
Transform options.

Specifies how the transform defined by the specified rotation should be used to modify the previous transform specified by the **Array** parameter. Possible values are:

TRANSFORM_REPLACE	The previous transform is discarded and replaced by the transform describing the specified rotation.
TRANSFORM_ADD	The previous transform is combined with a transform representing the specified rotation in the order (1) previous transform, (2) rotational transform. This option is most useful for incremental updates to transforms.

Angle *(ROF)* – input
Rotation angle.

The angle describing the rotation, measured counterclockwise from the x axis in degrees.

Center *(POINT)* – input
Center of rotation.

The point about which the rotation occurs.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_TRANSFORM_TYPE

Remarks

This call is a helper function which either applies a specified rotational component to an existing transform matrix, or replaces the matrix with one that represents the specified rotation alone.

The transform is specified as a one-dimensional array of 9 elements that are the elements of a 3-row by 3-column matrix ordered by rows. The order of the elements is:

```
Matrix                  Array

┌          ┐
│ a  b  0  │
│ c  d  0  │          (a,b,0,c,d,0,e,f,1)
│ e  f  1  │
└          ┘
```

Transforms act on the coordinates of primitives, so that a point with coordinates (x,y) is transformed to the point:

(a*x + c*y + e, b*x + d*y + f)

The transform can be used in any call following:

- GpiSetModelTransformMatrix
- GpiSetSegmentTransformMatrix
- GpiSetViewingTransformMatrix
- GpiSetDefaultViewMatrix.

Other similar helper functions are:

- GpiTranslate to apply a translation component
- GpiScale to apply a scaling component.

GpiSaveMetaFile — Save Metafile

SAA

This call saves a metafile to a file.

GpiSaveMetaFile (hmf, Filename, *Success***)**

Parameters

hmf *(HMF)* — input
Metafile handle.

Filename *(STRL)* — input
Filename.

The name of the file to which the metafile is to be saved. This name must be a valid external name.

It is an error if a file of this name exists already.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HMF
PMERR_DOSOPEN_FAILURE
PMERR_INSUFFICIENT_DISK_SPACE
PMERR_METAFILE_IN_USE
PMERR_TOO_MANY_METAFILES_IN_USE

Remarks

The metafile is deleted from storage; this means that the metafile handle is no longer valid.

The metafile may be reaccessed by GpiLoadMetaFile.

GpiSavePS — Save Presentation Space

This call saves various features of the presentation space on a LIFO (last-in-first-out) stack.

GpiSavePS (hps, *PSid*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

PSId *(LONG)* — return
Identifier of saved presentation space.

This may be used on a subsequent GpiRestorePS call. The identifier is equal to the depth of the saved presentation space on the save/restore stack, with 1 representing the base level:

≥1	Identifier of saved presentation space
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_NOT_IN_DRAW_MODE

Remarks

The stack is different from the one used to save attribute values (see GpiSetAttrMode), in a normal presentation space.

This call, and GpiRestorePS, can be used with a micro-presentation space, as well as a normal presentation space (in **draw** drawing mode only).

The presentation space itself is unchanged.

The following are saved:

- Current attributes
- Current transforms, viewing limits, and clip path
- Current position
- Reference to selected clip region
- Any loaded logical color table
- References to any loaded logical fonts
- References to the regions created on the associated device context.

The following are not saved:

- Draw controls
- Drawing mode
- Edit mode and attribute mode
- The visible region.

Programming Note: Only references to resources, rather than the actual resources (such as clip region, logical fonts, and regions) are copied by this call, so the actual resources must not be changed.

GpiSavePS – Save Presentation Space

SAA

This call is valid in an open segment bracket, but only if the drawing mode (see GpiSetDrawingMode) is set to **draw**. It can occur within an open element bracket. When it occurs within an open area or path bracket, the corresponding GpiRestorePS must take place before the bracket is closed.

If this call occurs during generation of a metafile, the drawing mode must be set to **draw** when the metafile is replayed.

GpiScale — Scale Matrix

This call applies a scaling component to a transform matrix.

GpiScale (hps, Array, Options, Scale, Center, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Array *(MATRIX)* — input/output
Transform matrix.

The elements of the transform, in row order. The first, second, fourth, and fifth elements are of type FIXED, and have an assumed binary point between the second and third bytes. Thus a value of 1.0 is represented by 65 536. Other elements are normal signed integers.

The third, sixth, and ninth elements must be 0, 0, and 1, respectively.

Options *(LONG)* — input
Transform options.

Specifies how the transform defined by the specified scaling should be used to modify the previous transform specified by the **Array** parameter. Possible values are:

TRANSFORM_REPLACE	The previous transform is discarded and replaced by the transform describing the specified scaling.
TRANSFORM_ADD	The previous transform is combined with a transform representing the specified scaling in the order (1) previous transform, (2) scaling transform. This option is most useful for incremental updates to transforms.

Scale *(ROF*2)* — input
Scale factors.

The first element of the array is the x scale factor, and the second is the y scale factor.

Scaling values outside the range −1 through +1 are not valid for subsequent use with presentation spaces that have a coordinate format (as set by the GpiCreatePS call) of GPIF_SHORT.

Center *(POINT)* — input
Center of scale.

The point about which the scale occurs.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_TRANSFORM_TYPE

GpiScale – Scale Matrix

Remarks

This call is a helper function which either applies a specified scaling component to an existing transform matrix, or replaces the matrix with one which represents the specified scaling alone.

The transform is specified as a one-dimensional array of 9 elements that are the elements of a 3-row by 3-column matrix ordered by rows. The order of the elements is:

```
Matrix                  Array

┌       ┐
│ a  b  0 │
│ c  d  0 │          (a,b,0,c,d,0,e,f,1)
│ e  f  1 │
└       ┘
```

Transforms act on the coordinates of primitives, so that a point with coordinates (x,y) is transformed to the point:

```
(a*x + c*y + e, b*x + d*y + f)
```

The transform can be used in any subsequent call to:

- GpiSetModelTransformMatrix
- GpiSetSegmentTransformMatrix
- GpiSetViewingTransformMatrix
- GpiSetDefaultViewMatrix.

Other similar helper functions are:

- GpiTranslate to apply a scaling component
- GpiRotate to apply a rotation component.

GpiSetArcParams — Set Arc Parameters

This call sets the current arc parameters.

GpiSetArcParams (hps, ArcParams, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

ArcParams *(ARCPARAM)* — input
Arc parameters.

This structure has four elements p, q, r, and s.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

The arc parameters p, q, r, and s, define the shape and orientation of a ellipse that is used for subsequent GpiPointArc, GpiFullArc, and GpiPartialArc calls. For GpiFullArc and GpiPartialArc, they also determine the direction of drawing, as follows:

- If p*q > r*s the direction is counterclockwise
- If p*q < r*s the direction is clockwise
- If p*q = r*s a straight line is drawn.

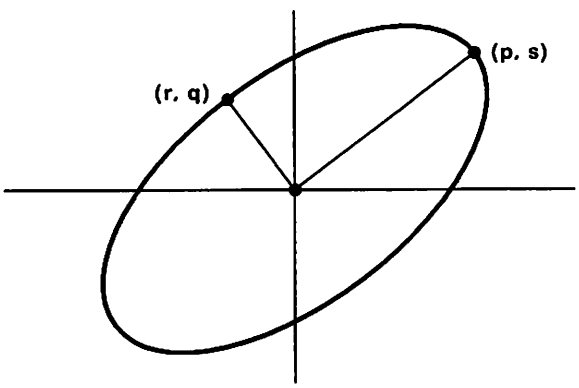


Figure 4-5. GpiSetArcParams Coordinate Points

For GpiFullArc and GpiPartialArc, these parameters also define the nominal size of the ellipse; this may be changed by using the multiplier.

For GpiPointArc, the size of the ellipse is determined by the three points specified on GpiPointArc.

GpiSetArcParams — Set Arc Parameters

SAA

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
```

With reference to Figure 4-5 on page 4-233, if p*r + s*q = 0, 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 must be used. The matrix must not be singular.

The initial default values of arc parameters (unless changed with GpiSetDefArcParams) are:

```
p = 1     r = 0
s = 0     q = 1
```

producing a unit circle; see Figure 4-6.

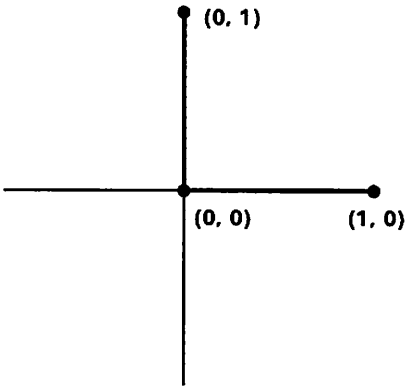


Figure 4-6. GpiSetArcParams Default Coordinates

Arc parameter transformation takes place in world coordinates. Any other non-square transformations in force change the shape of the figure accordingly.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the arc parameters is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSAP

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Arc Parameters

Element Type: OCODE_GPSAP

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Arc Parameters

GpiSetAttrMode — Set Attribute Mode

This call specifies the current attribute mode.

GpiSetAttrMode (hps, Mode, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Mode *(LONG)* — input
Attribute mode:

AM_PRESERVE	Preserve attributes
AM_NOPRESERVE	Do not preserve attributes.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_ATTR_MODE
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_DC_TYPE

Remarks

The attribute mode is used to specify whether a primitive attribute is to be preserved when set to a new value by a subsequent attribute setting call. The preserved value of an attribute can be restored using the GpiPop call. Any attributes that have been preserved in a called segment are automatically restored on return to the caller. However, the values of any attributes preserved in a *chained* segment that are not restored using GpiPop by the end of the segment are lost.

The following are affected:

- GpiSetArcParams
- GpiSetBackColor (applies individually by primitive type if GpiSetAttrs is used)
- GpiSetBackMix (applies individually by primitive type if GpiSetAttrs is used)
- GpiSetCharAngle
- GpiSetCharBox
- GpiSetCharDirection
- GpiSetCharMode
- GpiSetCharSet
- GpiSetCharShear
- GpiSetColor (applies individually by primitive type if GpiSetAttrs is used)
- GpiSetCurrentPosition
- GpiSetLineEnd
- GpiSetLineJoin
- GpiSetLineType
- GpiSetLineWidth
- GpiSetLineWidthGeom
- GpiSetMarker

- GpiSetMarkerBox
- GpiSetMarkerSet
- GpiSetMix (applies individually by primitive type if GpiSetAttrs is used)
- GpiSetModelTransformMatrix
- GpiSetPattern
- GpiSetPatternRefPoint
- GpiSetPatternSet
- GpiSetTag.

The value of the attribute mode before this call is issued, is AM_NOPRESERVE.

Attribute mode applies to attributes passed across the API using GpiSet... calls. The mode to use for a particular GpiSet... call is decided by the attribute mode current at the time the GpiSet... call is passed across the API. The mode may be changed at any time, and does not affect any attribute setting calls that have already been retained in the segment store.

Attribute mode only applies to individual GpiSet... calls (including GpiSetAttrs and calls such as GpiSetColor). It does not apply to any attribute setting calls passed across in bulk, such as GpiPutData, GpiElement, and GpiPlayMetaFile; these already indicate individually whether they should cause the attribute to be preserved.

Attribute mode cannot be set for GPIT_MICRO type presentation spaces associated with OD_METAFILE_NOQUERY type device contexts. In these cases, the presentation space behaves as if AM_NOPRESERVE is in operation.

GpiSetAttrs — Set Attributes

This call sets attributes for the specified primitive type.

GpiSetAttrs (hps, PrimType, AttrMask, DefMask, Attrs, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

PrimType *(LONG)* — input
Primitive type.

The primitive type for which attributes are to be set:

PRIM_LINE	Line and arc primitives
PRIM_CHAR	Character primitives
PRIM_MARKER	Marker primitives
PRIM_AREA	Area primitives
PRIM_IMAGE	Image primitives.

AttrMask *(BIT32)* — input
Attributes mask.

Each flag set indicates that either the corresponding flag in **DefMask** is set, or the **Attrs** buffer contains data for the corresponding attribute. If all of the flags in **AttrMask** are zero, the **Attrs** buffer address is not used.

Line attributes:

LBB_COLOR	Line color
LBB_MIX_MODE	Line mix
LBB_WIDTH	Line width
LBB_GEOM_WIDTH	Geometric line width
LBB_TYPE	Line type
LBB_END	Line end
LBB_JOIN	Line join.

Character attributes:

CBB_COLOR	Character color
CBB_BACK_COLOR	Character background color
CBB_MIX_MODE	Character mix
CBB_BACK_MIX_MODE	Character background mix
CBB_SET	Character set
CBB_MODE	Character mode
CBB_BOX	Character box
CBB_ANGLE	Character angle
CBB_SHEAR	Character shear
CBB_DIRECTION	Character direction.

Marker attributes:

MBB_COLOR	Marker color
MBB_BACK_COLOR	Marker background color
MBB_MIX_MODE	Marker mix
MBB_BACK_MIX_MODE	Marker background mix
MBB_SET	Marker set
MBB_SYMBOL	Marker symbol
MBB_BOX	Marker box.

Pattern attributes (areas):

ABB_COLOR	Area color
ABB_BACK_COLOR	Area background color
ABB_MIX_MODE	Area mix
ABB_BACK_MIX_MODE	Area background mix
ABB_SET	Pattern set
ABB_SYMBOL	Pattern symbol
ABB_REF_POINT	Pattern reference point.

Image attributes:

IBB_COLOR	Image color
IBB_BACK_COLOR	Image background color
IBB_MIX_MODE	Image mix
IBB_BACK_MIX_MODE	Image background mix.

DefMask *(BIT32)* — input
Defaults mask.

Each flag set (and for which **AttrMask** is also set) causes the corresponding attribute to be set to its default value.

Attrs *(BUNDLE)* — input
Attributes.

This is a structure containing the attribute value of each attribute for which the **AttrMask** flag is set (and which is not to be set to its default value), at the correct offset as specified below for the particular primitive type.

Primitive type	**Structure**
Line attributes	*LINEBUNDLE*
Character attributes	*CHARBUNDLE*
Marker attributes	*MARKERBUNDLE*
Pattern attributes (areas)	*AREABUNDLE*
Image attributes	*IMAGEBUNDLE*.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PRIMITIVE_TYPE
PMERR_UNSUPPORTED_ATTR
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR

PMERR_INV_LINE_END_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_ATTR
PMERR_INV_COORDINATE
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED

Remarks

Any attribute (for the specified primitive type) for which the appropriate flag is set in the **AttrMask** has its value updated:

- If the corresponding flag in **DefMask** is also set, the attribute is set to default.
- If the corresponding flag in **DefMask** is not set, the attribute is set to the value specified in the **Attrs** structure.

Any attribute for which the appropriate flag in **AttrMask** is not set is unchanged, regardless of the setting of the corresponding flag in **DefMask**.

The **DefMask** and **AttrMask** parameters each contain flags: each attribute of the primitive type in question is represented by one flag.

The data in the **Attrs** buffer consists of a structure of attribute data. The layout of the structure is fixed for each primitive type. Only data for attributes for which the flag is set in **AttrMask** (but not in **DefMask**) is inspected; any other data is ignored.

Programming Note: The buffer need be no longer than is necessary to contain the data for the highest offset attribute referenced.

If default values of attributes are required, they must be requested using the **DefMask**. The system does not recognize attribute values whose meaning would normally be "use default" in the **Attrs** buffer.

Where possible, invalid color values are detected by this call and cause an error (PMERR_INV_COLOR_ATTR or PMERR_INV_BACKGROUND_COL_ATTR) to be logged. Some invalid color values cannot be detected until draw time at which point the implementation optionally defaults them, or causes the above error to be logged. If an attempt to set an invalid value by this call is detected, none of the specified attributes is changed. Note, however, that some invalid attribute values (for example, colors and mixes) may not be detected until the attribute is used.

If this call occurs within a path bracket, it must only set:

- Line attributes, other than the geometric line width
- Character attributes, other than foreground or background colors or mixes
- Marker attributes, other than foreground or background colors or mixes.

The default values of attributes can be changed with GpiSetDefAttrs.

The attribute mode (see GpiSetAttrMode) determines whether the current values of the attributes are preserved. If they are, one "push" call is generated for each affected attribute, in the order in which the attributes are specified, in the appropriate xxxBUNDLE structure.

Graphic Elements and Orders

The element type depends on the **PrimType** parameter.

For each element, the "Set" orders are generated, if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE, and the "Push and Set" orders if it is AM_PRESERVE. In either instance, a particular order is generated only if the corresponding attribute is being set with this call, as specified on the **Attrs** parameter.

Element Type: ETYPE_LINEBUNDLE

Generated if **PrimType** is PRIM_LINE.

As many as required of the following are generated if the attribute mode is AM_NOPRESERVE:

Order: Set Individual Attribute

One of these for each of LBB_COLOR and LBB_MIX_MODE, as required.

Order: Set Fractional Line Width

LBB_WIDTH, as required.

Order: Set Stroke Line Width

LBB_GEOM_WIDTH, as required.

Order: Set Line Type

LBB_TYPE, as required.

Order: Set Line End

LBB_END, as required.

Order: Set Line Join

LBB_JOIN, as required.

As many as required of the following are generated if the attribute mode is AM_PRESERVE:

Order: Push and Set Individual Attribute

One of these for each of LBB_COLOR and LBB_MIX_MODE, as required.

Order: Push and Set Fractional Line Width

LBB_WIDTH, as required.

Order: Push and Set Stroke Line Width

LBB_GEOM_WIDTH, as required.

Order: Push and Set Line Type

LBB_TYPE, as required.

Order: Push and Set Line End

LBB_END, as required.

Order: Push and Set Line Join

LBB_JOIN, as required.

Element Type: ETYPE_CHARBUNDLE

Generated if **PrimType** is PRIM_CHAR.

As many as required of the following are generated if the attribute mode is AM_NOPRESERVE:

Order: Set Individual Attribute

One of these for each of CBB_COLOR, CBB_BACK_COLOR, CBB_MIX_MODE, and CBB_BACK_MIX_MODE, as required.

Order: Set Character Set

CBB_SET

Order: Set Character Precision

CBB_MODE

Order: Set Character Cell

CBB_BOX

Order: Set Character Angle

CBB_ANGLE

Order: Set Character Shear

CBB_SHEAR

Order: Set Character Direction

CBB_DIRECTION

As many as required of the following are generated if the attribute mode is AM_PRESERVE:

Order: Push and Set Individual Attribute

One of these for each of CBB_COLOR, CBB_BACK_COLOR, CBB_MIX_MODE, and CBB_BACK_MIX_MODE, as required.

Order: Push and Set Character Set

CBB_SET

Order: Push and Set Character Precision

CBB_MODE

Order: Push and Set Character Cell

CBB_BOX

Order: Push and Set Character Angle

CBB_ANGLE

Order: Push and Set Character Shear

CBB_SHEAR

Order: Push and Set Character Direction

CBB_DIRECTION

Element Type: ETYPE_MARKERBUNDLE

Generated if **PrimType** is PRIM_MARKER.

As many as required of the following are generated if the attribute mode is AM_NOPRESERVE:

Order: Set Individual Attribute

One of these for each of MBB_COLOR, MBB_BACK_COLOR, MBB_MIX_MODE, and MBB_BACK_MIX_MODE, as required.

Order: Set Marker Set

MBB_SET

Order: Set Marker Symbol

MBB_SYMBOL

Order: Set Marker Cell

MBB_BOX

As many as required of the following are generated if the attribute mode is AM_PRESERVE:

Order: Push and Set Individual Attribute

One of these for each of MBB_COLOR, MBB_BACK_COLOR, MBB_MIX_MODE, and MBB_BACK_MIX_MODE, as required.

Order: Push and Set Marker Set

MBB_SET

Order: Push and Set Marker Symbol

MBB_SYMBOL

Order: Push and Set Marker Cell

MBB_BOX

Element Type: ETYPE_AREABUNDLE

Generated if **PrimType** is PRIM_AREA.

As many as required of the following are generated if the attribute mode is AM_NOPRESERVE:

Order: Set Individual Attribute

One of these for each of ABB_COLOR, ABB_BACK_COLOR, ABB_MIX_MODE, and ABB_BACK_MIX_MODE, as required.

Order: Set Pattern Set

ABB_SET

Order: Set Pattern Symbol

ABB_SYMBOL

Order: Set Pattern Reference Point

ABB_REF_POINT

As many as required of the following are generated if the attribute mode is AM_PRESERVE:

Order: Push and Set Individual Attribute

One of these for each of ABB_COLOR, ABB_BACK_COLOR, ABB_MIX_MODE, and ABB_BACK_MIX_MODE, as required.

Order: Push and Set Pattern Set

ABB_SET

Order: Push and Set Pattern Symbol

ABB_SYMBOL

Order: Push and Set Pattern Reference Point

ABB_REF_POINT

Element Type: ETYPE_IMAGEBUNDLE

Generated if **PrimType** is PRIM_IMAGE.

As many as required of the following are generated if the attribute mode is AM_NOPRESERVE:

Order: Set Individual Attribute

One of these for each of IBB_COLOR, IBB_BACK_COLOR, IBB_MIX_MODE, and IBB_BACK_MIX_MODE, as required.

As many as required of the following are generated if the attribute mode is AM_PRESERVE:

Order: Push and Set Individual Attribute

One of these for each of IBB_COLOR, IBB_BACK_COLOR, IBB_MIX_MODE, and IBB_BACK_MIX_MODE, as required.

GpiSetBackColor — Set Background Color

SAA

This call sets the current background color index attribute, for each individual primitive type, to the specified value.

GpiSetBackColor (hps, Color, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Color *(LONG)* — input
Background color:

For a loadable color table, values 0 through n correspond to the color index (or RGB) values.

CLR_FALSE	All color planes are 0s.
CLR_TRUE	All color planes are 1s.
CLR_DEFAULT	Set to default value. This is the natural background color for the device. For a display, it is the default window color (SYSCLR_WINDOW: see WinSetSysColors). For a printer, it is the paper color. The default can be changed by setting new system colors from the control panel for the display, or by selecting a paper color for a printer (if allowed by the device driver), or set explicitly with GpiSetDefAttrs.
CLR_WHITE	White (default color table, or index=RGB loaded color table). For a loaded, realized, color table, it is the nearest available color to white.
CLR_BLACK	Black (default color table, or index=RGB loaded color table). For a loaded, realized, color table, it is the nearest available color to black.
CLR_BACKGROUND	Reset color, used by GpiErase. This is the natural background color for the device. For a display, it is the default window color (SYSCLR_WINDOW: see WinSetSysColors) for the default color table. For a printer, it is the paper color. For a loaded color table, it is color index 0. For an RGB color table, it is color 000000 (black).
CLR_BLUE	Blue (default color table).
CLR_RED	Red (default color table).
CLR_PINK	Pink (default color table).
CLR_GREEN	Green (default color table).
CLR_CYAN	Cyan (default color table).
CLR_YELLOW	Yellow (default color table).
CLR_NEUTRAL	Neutral (default color table). A device-dependent color, that for the default color table provides a contrasting color to CLR_BACKGROUND. For a display, it is the default window text color (SYSCLR_WINDOWTEXT: see WinSetSysColors). For a printer, it is a color that contrasts with the paper color. For a loaded color table, it is color index 7; in RGB mode it is interpreted as color 000007.
CLR_DARKGRAY	Dark gray (default color table).
CLR_DARKBLUE	Dark blue (default color table).
CLR_DARKRED	Dark red (default color table).
CLR_DARKPINK	Dark pink (default color table).
CLR_DARKGREEN	Dark green (default color table).
CLR_DARKCYAN	Dark cyan (default color table).
CLR_BROWN	Brown (default color table).
CLR_PALEGRAY	Pale gray (default color table).

For a loadable color table, values 0 through n correspond to the color index (or RGB) values.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_BACKGROUND_COL_ATTR

Remarks

Note that if the background mix is BM_LEAVEALONE (the default setting), the background color is not seen.

An attempt to set a negative color value, other than one for which a constant is defined, causes the error PMERR_INV_COLOR_ATTR to be logged. Other color values are allowed, although an error is generated when the color value is needed for drawing if it is invalid for the color table in use at that time (see GpiCreateLogColorTable).

For details of how colors are handled on monochrome devices, see GpiCreateLogColorTable.

The attribute mode determines whether the current value of the background color attribute is preserved. If it is, the values of the background color attribute, for each primitive type, are preserved, and a single GpiPop call restores them.

This call must not be used within a path or area bracket.

Graphic Elements and Orders

Element Type: OCODE_GSBICOL

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Background Indexed Color

Element Type: OCODE_GPSBICOL

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Background Indexed Color

GpiSetBackMix — Set Background Mix

SAA

This call sets the current background mix attribute for each individual primitive type.

GpiSetBackMix (hps, MixMode, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

MixMode *(LONG)* — input
Background-mix mode.

Background mixes marked with an asterisk (*) are mandatory for all devices.

The currently associated device supports any of the mixes specified as supported in DevQueryCaps (CAPS_BACKGROUND_MIX_SUPPORT) Any other valid mixes can be supported for some primitive types, but otherwise result in BM_LEAVEALONE. An error is raised only if the value specified is not one of those listed.

For more information on mixing, see GpiSetMix.

BM_DEFAULT	The default value (BM_LEAVEALONE unless changed with GpiSetDefAttrs).
BM_OR	Logical-OR.
BM_OVERPAINT	The background of the primitive takes precedence over whatever is underneath. (*)
BM_XOR	Exclusive-OR.
BM_LEAVEALONE	The background of the primitive has no effect on what is underneath. (*)

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_BACKGROUND_COL_ATTR

Remarks

Controls the way that the background color of a primitive is combined with the color of any primitive that it overlaps.

These primitives are affected by the background mix attribute:

Areas	The background of an area is defined to be every pel within the area that is not set by the shading pattern.
Text	The background of a character is the complete character box.
Images	For an image, the background is every pel within the image that is not set.
Markers	The background of a marker is the complete marker box.

Programming Note: When the background mix is BM_LEAVEALONE (initial default) the background color is not seen.

The attribute mode determines whether the current value of the background mix attribute is preserved. If it is, the values of the background mix attribute for each primitive type are preserved, and a single GpiPop call restores them.

This function should not be used within a path or area bracket.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles (see "Metafile Restrictions" on page D-1).

Graphic Elements and Orders

Element Type: OCODE_GSBMX

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Background Mix

Element Type: OCODE_GPSBMX

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Background Mix

GpiSetBitmap — Set Bit Map

SAA

This call sets a bit map as the currently selected bit map in a memory device context.

GpiSetBitmap (hps, hbm, *Old*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

hbm *(HBITMAP)* — input
Handle of the bit map to be set.

A null handle causes the currently selected bit map, in the associated device, to be freed.

It is an error if the bit map is currently tagged for area shading (see GpiSetBitmapId).

Old *(HBITMAP)* — return
Old bit-map handle:

NULL	Correct (null handle)
HBM_ERROR	Error
Otherwise	Old bit-map handle.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HBITMAP
PMERR_BITMAP_IN_USE
PMERR_INCOMPATIBLE_BITMAP
PMERR_HBITMAP_BUSY

Remarks

The specified presentation space must be currently associated with a memory device context. The device context can represent a different physical device from the one that the bit map originally loaded or created, providing its format is convertible to one supported on the new device. This is ensured when one of the standard formats is being used.

If a bit map is already current in the device context, the handle of this bit map is returned, before the new bit map is selected.

It is an error if the new bit map is already selected as the current bit map in any device context.

GpiSetBitmapBits — Set Bit-Map Bits

This call transfers bit-map data from application storage to a bit map.

GpiSetBitmapBits (hps, ScanStart, Scans, Buffer, InfoTable, *ScansSet***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

ScanStart *(LONG)* — input
Line number

Scan-line number at which the data transfer is to start, counting from zero as the bottom line.

Scans *(LONG)* — input
Number of scan lines to be transmitted.

Buffer *(BUFFER)* — input
Bit-map data buffer.

Address in application storage from which the bit-map data is to be copied.

InfoTable *(BITMAPINFO)* — input
Bit-map information table.

ScansSet *(LONG)* — return
Number of scan lines actually set:

≥0	Number of scan lines actually set
GPI_ALTERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_INFO_TABLE
PMERR_NO_BITMAP_SELECTED
PMERR_INV_SCAN_START
PMERR_INCORRECT_DC_TYPE

Remarks

The presentation space must be currently associated with a memory device context that has a bit map currently selected.

Programming Note: This call does not set bits directly to any other kind of device.

If the format of the supplied bit map does not match that of the device, it is converted, using the supplied bit-map information table. The standard bit-map formats are supported, plus any known to be supported by the device; see GpiQueryDeviceBitmapFormats.

GpiSetBitmapDimension — Set Bit-Map Dimension

This call associates a width and height with a bit map, in units of 0.1 millimeter.

GpiSetBitmapDimension (hbm, BitmapDimension, *Success*)

Parameters

hbm *(HBITMAP)* — input
Bit-map handle.

BitmapDimension *(SIZEROL)* — input
Width and height of bit map.

The width and height, respectively, of the bit map in units of 0.1 millimeter.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HBITMAP
PMERR_HBITMAP_BUSY

Remarks

The values set are not used internally by the system, but are retained with the bit map and can be retrieved with GpiQueryBitmapDimension.

GpiSetBitmapId — Set Bit-Map Identifier

This call tags a bit map with a local identifier, so that it can be used as a pattern set, containing a single member.

GpiSetBitmapId (hps, hbm, Lcid, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

hbm *(HBITMAP)* — input
Bit-map handle.

The bit map must not be currently selected into a device context (see GpiSetBitmap).

Lcid *(LONG)* — input
Local identifier with which the bit map is to be tagged.

Valid values are in the range 1 through 254.

It is an error if the local identifier is already used to refer to a font or bit map.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HBITMAP
PMERR_INV_SETID
PMERR_SETID_IN_USE
PMERR_BITMAP_IN_USE
PMERR_HBITMAP_BUSY

Remarks

To use the bit map for area shading (or as the pattern in a GpiBitBlt or GpiWCBitBlt operation), a GpiSetPatternSet must be issued with the specified local identifier.

Any bit map of a format supported by the device can be specified. However, it may be simplified before use (see GpiSetPatternSet).

GpiDeleteSetId can subsequently be used to release the tag.

GpiSetCharAngle – Set Character Angle

SAA

This call specifies the angle of the baseline for the characters in a string, as a relative vector.

GpiSetCharAngle (hps, Angle, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Angle *(GRADIENT)* – input
Baseline angle.

The baseline angle is defined in terms of the relative coordinates of the point **Angle** (**ax**,**ay**).

If both **ax** and **ay** are **0**, the character angle is reset to the default value. This default value is (1,0), unless changed with GpiSetDefAttrs.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

The coordinates of the point **Angle** specify integer values for the coordinates of the end of a line starting at the origin (0,0); the base line for subsequent character strings is parallel to this line.

The effect of the baseline angle attribute depends on the value of the character mode attribute (see GpiSetCharMode), and whether the current font is an outline or a raster font, as described below.

When the character mode is set to CM_MODE1, and the current font is a raster font, the character angle can be ignored.

When the character mode is set to CM_MODE2, and the current font is a raster font, the angle is used to determine the position of each character, but the orientations of characters within the character box may not be affected by changes in character angle. If this is so, the characters are positioned so that the lower left-hand corners of the character definitions are placed at the lower left-hand corners of the character boxes after all transforms have been applied. This is illustrated in Figure 4-7 on page 4-253.

For illustrative purposes the figure assumes that all character reference points are at their bottom left hand corner.

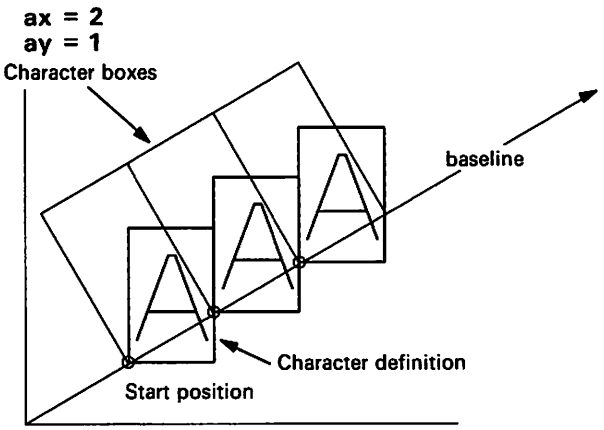


Figure 4-7. Character Angle and Mode-2 Text Positioning

When the character mode is set to CM_MODE3, or when the current font is an outline font, the angle is observed accurately, and the character boxes are rotated to be normal (perpendicular) to the character baseline. If the world coordinate 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 call must not be issued in an area bracket.

The attribute mode determines whether the current value of the baseline angle attribute is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSCA

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Character Angle

Element Type: OCODE_GPSCA

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Character Angle

GpiSetCharBox — Set Character Box

SAA

This call sets the current character-box attribute to the specified value.

GpiSetCharBox (hps, Box, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Box *(SIZEROF)* — input
Character-box size in world coordinates.

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 becomes a line. If both are zero, the string is drawn as a single point.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

The parameter **Box** specifies values for the width and height of a character box in world coordinates.

Whether these values are used when character strings are drawn depends on the type of font being used (raster or outline), and on the value of the character mode attribute (see the GpiSetCharMode call).

For raster fonts, where the character box is used only for positioning in character mode CM_MODE2, the character box width corresponds to the **eminc** font metric (see *FONTMETRICS*). For proportionally-spaced raster fonts, the effective spacing for a given character is the character box width, scaled by the ratio of that character's width to **eminc**.

For outline fonts, the characters are drawn by scaling the **xdeviceres** and **ydeviceres** font metrics (see *FONTMETRICS*) to the character box width and height, respectively. The effective width of each character is the character box width, scaled by the ratio of that character's width to **xdeviceres**.

To cause characters of a given point-size to be generated using an outline font, establish a world coordinate space with specific metrics (for example, specify PU_TWIPS on GpiCreatePS), and set the character box height to the required point size. Since **xdeviceres** and **ydeviceres** are normally equal, the character box width should also be set equal to the height, if characters are required with the same aspect ratio as defined in the font (assuming that world coordinate space is *isotropic*).

The initial default value of the character box is the device-coordinates value returned by DevQueryCaps (CAPS_GRAPHICS_CHAR_WIDTH and CAPS_GRAPHICS_CHAR_HEIGHT), for the currently associated device, converted to page coordinates.

Programming Note: In general the initial default value is rectangular, and to avoid character distortion, the character box should normally be set explicitly to be square if an outline font might be used (assuming that world coordinate space is isotropic).

The default value can be changed with GpiSetDefAttrs.

GpiSetCharBox must not be issued in an area bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the character-box size attribute is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSCC

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Character Cell

Element Type: OCODE_GPSCC

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Character Cell

GpiSetCharDirection — Set Character Direction

SAA

This call determines the direction in which the characters in a string are drawn relative to the baseline.

GpiSetCharDirection (hps, Direction, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Direction *(LONG)* — input
Character direction:

CHDIRN_DEFAULT The default; the same as CHDIRN_LEFTRIGHT (unless changed with GpiSetDefAttrs).

CHDIRN_LEFTRIGHT Character boxes are arranged parallel to, and in the same direction as, the baseline. This is the usual convention for Roman text.

CHDIRN_TOPBOTTOM Character boxes are arranged in columns directed 90 degrees *clockwise* from the baseline. This is the usual convention for Chinese characters. This option can be used for drawing Roman text vertically (a y-axis title on a graph, for example). The reference point within the character definition is at the center of the character, along the x direction, in this case.

CHDIRN_RIGHTLEFT Character boxes are arranged parallel to, but in the reverse of, the baseline direction. This is the usual convention for Arabic text.

CHDIRN_BOTTOMTOP Character boxes are arranged in columns directed 90 degrees *counterclockwise* from the baseline. The reference point within the character definition is at the center of the character, along the x direction, in this case.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CHAR_DIRECTION_ATTR

Remarks

This function must not be issued in an area bracket. The attribute mode determines whether the current value of the character direction attribute is preserved. This diagram shows how the origin of characters changes when the direction changes:

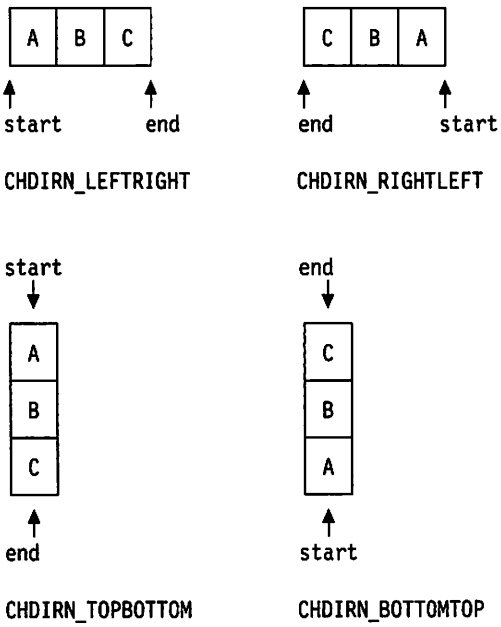


Figure 4-8. Character Drawing Directions and Character Box Origins

Graphic Elements and Orders

Element Type: OCODE_GSCD

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Character Direction

Element Type: OCODE_GPSCD

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Character Direction

GpiSetCharMode — Set Character Mode

SAA

This call controls the character mode used when drawing a character string.

GpiSetCharMode (hps, Mode, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Mode *(LONG)* — input
Character mode:

CM_DEFAULT The default; the same as CM_MODE1 (unless changed with GpiSetDefAttrs).

CM_MODE1 The font selected by means of GpiSetCharSet can be either a raster font or an outline font.

If it is a raster font, the position of characters after the first character is determined by the font metrics information, and the character direction attribute. If it is an outline font, the behavior is as if the character mode is CM_MODE3.

CM_MODE2 The font selected by means of GpiSetCharSet can be either a raster font or an outline font.

If it is a raster font, the position of characters after the first character is determined by the font metrics information, and certain character attributes, namely, GpiSetCharAngle, GpiSetCharBox, GpiSetCharDirection, and GpiSetCharShear (the latter is relevant only for character directions of CHDIRN_TOPBOTTOM and CHDIRN_BOTTOMTOP). If it is an outline font, the behavior is as if the character mode is CM_MODE3.

CM_MODE3 All character attributes are used for positioning (together with the positioning information in the font), and for scaling, rotating, and shearing the characters.

The font selected by means of GpiSetCharSet must be an outline font; an error is raised if an attempt is made to draw a character string in CM_MODE3, and the selected font is a raster font.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CHAR_MODE_ATTR

Remarks

The value of the **Mode** parameter controls whether the character attributes affect the positioning of characters, as follows:

Table 4-1. Use of Character Attributes in each Character Mode

Character Mode	Font Type	Character Attributes (Angle, Shear, and Box)
Mode 1	Raster	Ignored
	Outline	Used
Mode 2	Raster	Attribute information is used to position characters; characters are not sheared, rotated, or scaled.
	Outline	Used
Mode 3	Raster	An error is raised when an attempt is made to draw a character string.
	Outline	Used

This call must not be issued in an area bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the character-mode attribute is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSCR

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Character Precision

Element Type: OCODE_GPSCR

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Character Precision

GpiSetCharSet – Set Character Set

SAA

This call sets the current value of the character-set attribute.

GpiSetCharSet (hps, lcid, *Success***)**

Parameters

hps *(HPS)* – input
Presentation-space handle.

lcid *(LONG)* – input
Character-set local identifier:

LCID_DEFAULT Default (can be set explicitly with GpiSetDefAttrs).
1 – 254 Identifies a logical font.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CHAR_SET_ATTR
PMERR_HUGE_FONTS_NOT_SUPPORTED

Remarks

This call must not be issued in an area bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the character-set attribute is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSCS

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Character Set

Element Type: OCODE_GPSCS

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Character Set

GpiSetCharShear — Set Character Shear

This call sets the character-shear attribute.

GpiSetCharShear (hps, Angle, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Angle *(POINT)* — input
Character shear vector.

With reference to Figure 4-9 on page 4-262, the shear angle is defined in terms of the relative coordinates of the point **Angle** (ax,ay).

If ax is 0 and ay is 1 (initial default), "upright" characters result. If ax and ay are both positive or both negative, the characters slope from bottom-left to top-right. If ax and ay are of opposite signs, the characters slope from top-left to bottom-right. No character inversion ever takes place as a result of a shear alone.

Usually, it is an error to specify 0 for ay, because this implies an "infinite" shear. However, if both ax and ay are 0, the attribute is set to the default value. This can be changed from the initial default with GpiSetDefAttrs.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_COORDINATE

Remarks

The coordinates of the point **Angle** (ax,ay), specify integer values that identify the end coordinates of a line originating at (0,0) (see Figure 4-9 on page 4-262). 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 (which may itself be rotated).

Whether this attribute is used when character strings are drawn depends on the type of font being used (raster or outline), and on the value of the character mode attribute (see GpiSetCharMode). If it is used, then with character directions of CHDIRN_TOPBOTTOM and CHDIRN_BOTTOMTOP (see GpiSetCharDirection) the whole string is tilted by the shear angle, in addition to the individual characters being sheared if the current font is an outline font.

This call must not be issued in an area bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the character shear attribute is preserved.

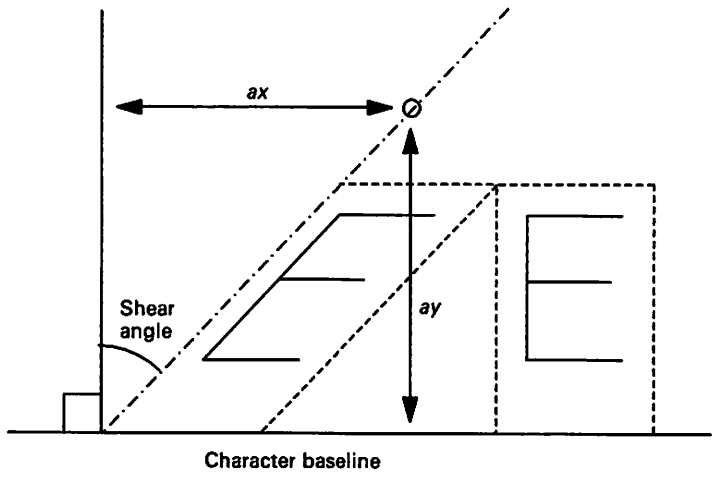


Figure 4-9. Character Shear

Graphic Elements and Orders

Element Type: OCODE_GSCH

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Character Shear

Element Type: OCODE_GPSCH

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Character Shear

GpiSetClipPath — Set Clip Path

This call selects a path as the current clip path.

GpiSetClipPath (hps, Path, Options, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Path *(LONG)* — input
Path control flag.

- **0** The current clip path stops being the current clip path; the current clip path is to be reset to an infinite one (that is, no clipping)
- **1** The path that has been defined is to be intersected with the current clip path.

Options *(LONG)* — input
Options.

This contains fields of option bits. For each field, one value should be selected (unless the default is suitable). These values can then be ORed together to generate the parameter.

How to construct the path interior (see also GpiBeginArea):

SCP_ALTERNATE Construct interior in alternate mode
SCP_WINDING Construct interior in winding mode. This value must be selected if the path has been modified using GpiModifyPath.

The default is SCP_ALTERNATE.

Relationship to current clip path:

SCP_AND Intersect the specified path with the current clip path. This must be specified if **Path** is 1.
SCP_RESET Reset the clip path to an infinite clip path. This must be specified or defaulted if **Path** is 0.

The default is SCP_RESET.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PATH_ID
PMERR_INV_CLIP_PATH_OPTIONS
PMERR_PATH_UNKNOWN

Remarks

The clip path (bound in device coordinates when the path is defined) is used for all subsequent drawing.

Any open figures within the path are closed automatically.

The boundaries of the area defined by the path are considered to be part of the interior, so that a point on the boundary is not clipped.

GpiSetClipPath — Set Clip Path

SAA

The clip path is reset to no clipping (no path selected) at the start of a root segment (subject to the fast chaining segment attribute), or when a GpiResetPS call is issued.

After a path is selected as the clip path, it cannot be reused for any other purpose. When it is superseded as the clip path, it is discarded.

Graphic Elements and Orders

Element Type: OCODE_GSCPTH

Order: Select Clip Path

GpiSetClipRegion — Set Clip Region

This call defines the region to be used for clipping, when any drawing takes place through the specified presentation space.

GpiSetClipRegion (hps, hrgn, Old, *Complexity*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The presentation space must be currently associated with a device context of the correct device class (defined when the region is first created).

hrgn *(HRGN)* — input
Region handle.

If **hrgn** is null, the clipping region is set to no clipping (its initial state).

Old *(HRGN)* — output
Old region handle (if any):

HRGN_ERROR	Error
NULL	Null handle (no region selected)
Otherwise	Old region handle.

Complexity *(LONG)* — return
Complexity of clipping/error indicator:

The clipping complexity information includes the combined effects of:

- Clip path
- Viewing limits
- Graphics field
- Clip region
- Visible region (windowing considerations).

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HRGN
PMERR_HRGN_BUSY

GpiSetClipRegion — Set Clip Region

SAA

Remarks

While a region is in use as a clip region, the calls GpiOffsetClipRegion, GpiExcludeClipRectangle and GpiIntersectClipRectangle cause it to be changed. These changes persist after the region has been deselected. The clip region cannot, however, be used for any other region operations, nor can it be selected into any other presentation space as a clipping region, until it is deselected.

The coordinates of the region are taken to be device coordinates within the device context.

The previous clip region, if any, is converted to a region, and a handle to it is returned. This can be used in a subsequent GpiSetClipRegion to reinstate the same clipping as before. If no clip region exists, a null handle is returned.

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiSetColor — Set Color

This call sets the current value of the color attribute for each of the individual primitive types.

GpiSetColor (hps, Color, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Color *(LONG)* — input
Color:

CLR_FALSE	All color planes are 0s.
CLR_TRUE	All color planes are 1s.
CLR_DEFAULT	Set to default value. This is a device-dependent color which, for the default color table, provides a contrasting color to CLR_BACKGROUND For a display, it is the default window text color (SYSCLR_WINDOWTEXT: see WinSetSysColors). For a printer, it is a color that contrasts with the paper color. The default can be changed by setting new system colors from the control panel for the display, or by selecting a paper color for a printer, if allowed by the device driver. It can also be set explicitly, using GpiSetDefAttrs.
CLR_WHITE	White (default color table, or index=RGB loaded color table). For a loaded, realized, color table it is the nearest available color to white.
CLR_BLACK	Black (default color table, or index=RGB loaded color table). For a loaded, realized, color table, it is the nearest available color to black.
CLR_BACKGROUND	Reset color, used by GpiErase. This is the natural background color for the device. For a display, it is the default window color (SYSCLR_WINDOW: see WinSetSysColors) for the default color table. For a printer, it is the paper color. For a loaded color table, it is color index 0. For an RGB color table, it is color 000000 (black).
CLR_BLUE	Blue (default color table).
CLR_RED	Red (default color table).
CLR_PINK	Pink (default color table).
CLR_GREEN	Green (default color table).
CLR_CYAN	Cyan (default color table).
CLR_YELLOW	Yellow (default color table).
CLR_NEUTRAL	Neutral (default color table). A device-dependent color, that for the default color table provides a contrasting color to CLR_BACKGROUND. For a display, it is the default window text color (SYSCLR_WINDOWTEXT: see WinSetSysColors). For a printer, it is a color that contrasts with the paper color. For a loaded color table, it is color index 7; in RGB mode it is interpreted as color 000007.
CLR_DARKGRAY	Dark gray (default color table).
CLR_DARKBLUE	Dark blue (default color table).
CLR_DARKRED	Dark red (default color table).
CLR_DARKPINK	Dark pink (default color table).
CLR_DARKGREEN	Dark green (default color table).
CLR_DARKCYAN	Dark cyan (default color table).
CLR_BROWN	Brown (default color table).
CLR_PALEGRAY	Pale gray (default color table).
	For a loadable color table, values 0 through n correspond to the color index (or RGB) values.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COLOR_ATTR

Remarks

The current values for each primitive type are updated. The attribute mode (see GpiSetAttrMode) determines whether the current values of the individual color attributes are preserved. If so, they are restored by a single GpiPop call.

An attempt to set a negative color value, other than one for which a constant is defined, causes the error PMERR_INV_COLOR_ATTR to be logged. Other color values are allowed, although an error is generated when the color value is needed for drawing if it is not valid for the color table in use at that time (see GpiCreateLogColorTable).

For details of how colors are handled on monochrome devices, see GpiCreateLogColorTable.

Graphic Elements and Orders

Element Type: OCODE_GSICOL

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Indexed Color

Element Type: OCODE_GPSICOL

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Indexed Color

GpiSetCp — Set Code Page

This call selects the code page id to be used for graphics characters for the default font.

GpiSetCp (hps, CodePage, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

CodePage *(USHORT)* — input
Code-page id.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_CODEPAGE

Remarks

Any code page that is defined in CONFIG.SYS or is a supported EBCDIC code page can be selected.

The list of available code pages is returned by WinQueryCpList.

When a GPI presentation space is first created, the code page in force is that defined by the process code page.

If this call occurs within a path definition when the drawing mode (see GpiSetDrawingMode) is **retain** or **draw-and-retain**, its effect is not stored with the definition.

Note: There are restrictions on the use of this call when creating SAA-conforming metafiles (see "Metafile Restrictions" on page D-1).

GpiSetCurrentPosition — Set Current Position

SAA

This call sets the current position to the specified point.

GpiSetCurrentPosition (hps, Point, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — input
New value of current position.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

This call also has the effect of resetting the position within a line-type sequence, and, if within an area, of starting a new closed figure and causing any previous one to be closed if necessary.

This call is equivalent to the GpiMove call except that, if the current attribute mode is AM_PRESERVE (see GpiSetAttrMode), the current position is saved before being set to the new value, so that it can be restored using the GpiPop call.

Graphic Elements and Orders

Element Type: OCODE_GSCP

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Current Position

Element Type: OCODE_GPSCP

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Current Position

GpiSetDefArcParams — Set Default Arc Parameters

This call specifies the default values of the arc parameters (see GpiSetArcParams).

GpiSetDefArcParams (hps, ArcParams, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

ArcParams *(ARCPARAM)* — input
Default arc parameters.

This structure has four elements p, q, r, and s.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

The arc parameters are reset to their default values at the following times:

- When the presentation space is associated with a device context (see GpiAssociate).
- When GpiResetPS is issued.
- When drawing of a chained segment begins or ends (see GpiOpenSegment and GpiCloseSegment for more details).

The initial default values of the arc parameters, when the presentation space is first created, are:

```
p = 1     r = 0
s = 0     q = 1
```

The default values can be changed by GpiSetDefArcParams. Changing the default values has an immediately effect on the current arc parameters, if these are currently set to the default value.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiSetDefAttrs — Set Default Attributes

This call sets the default values of attributes for the specified primitive type.

GpiSetDefAttrs (hps, PrimType, AttrMask, Attrs, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

PrimType *(LONG)* — input
Primitive type.

The primitive type for which default attributes are to be set:

PRIM_LINE	Line and arc primitives
PRIM_CHAR	Character primitives
PRIM_MARKER	Marker primitives
PRIM_AREA	Area primitives
PRIM_IMAGE	Image primitives.

AttrMask *(BIT32)* — input
Attributes mask.

Each flag that is set indicates that the **Attrs** buffer contains data for the corresponding attribute. If all of the flags in **AttrMask** are zero, the **Attrs** buffer address is not used.

Line attributes:

LBB_COLOR	Line color
LBB_MIX_MODE	Line mix
LBB_WIDTH	Line width
LBB_GEOM_WIDTH	Geometric line width
LBB_TYPE	Line type
LBB_END	Line end
LBB_JOIN	Line join.

Character attributes:

CBB_COLOR	Character color
CBB_BACK_COLOR	Character background color
CBB_MIX_MODE	Character mix
CBB_BACK_MIX_MODE	Character background mix
CBB_SET	Character set
CBB_MODE	Character mode
CBB_BOX	Character box
CBB_ANGLE	Character angle
CBB_SHEAR	Character shear
CBB_DIRECTION	Character direction.

Marker attributes:

MBB_COLOR	Marker color
MBB_BACK_COLOR	Marker background color
MBB_MIX_MODE	Marker mix
MBB_BACK_MIX_MODE	Marker background mix
MBB_SET	Marker set
MBB_SYMBOL	Marker symbol
MBB_BOX	Marker box.

Pattern attributes (areas):

ABB_COLOR	Area color
ABB_BACK_COLOR	Area background color
ABB_MIX_MODE	Area mix
ABB_BACK_MIX_MODE	Area background mix
ABB_SET	Pattern set
ABB_SYMBOL	Pattern symbol
ABB_REF_POINT	Pattern reference point.

Image attributes:

IBB_COLOR	Image color
IBB_BACK_COLOR	Image background color
IBB_MIX_MODE	Image mix
IBB_BACK_MIX_MODE	Image background mix.

Attrs *(BUNDLE)* — input
Default attribute values.

This is a structure containing default attribute values for each attribute for which the **AttrMask** flag is set, at the correct offset as specified below for the particular primitive type.

Line attributes: **Attrs** consists of a *LINEBUNDLE* structure.

Character attributes: **Attrs** consists of a *CHARBUNDLE* structure.

Marker attributes: **Attrs** consists of a *MARKERBUNDLE* structure.

Pattern attributes (areas): **Attrs** consists of a *AREABUNDLE* structure.

Image attributes: **Attrs** consists of a *IMAGEBUNDLE* structure.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PRIMITIVE_TYPE
PMERR_UNSUPPORTED_ATTR
PMERR_INV_COLOR_ATTR
PMERR_INV_BACKGROUND_COL_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_MARKER_SYMBOL_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_ATTR
PMERR_INV_COORDINATE
PMERR_UNSUPPORTED_ATTR_VALUE
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED

GpiSetDefAttrs – Set Default Attributes

Remarks

Attributes are reset to their default values at the following times:

- When the presentation space is associated with a device context (see GpiAssociate).
- When GpiResetPS is issued.
- When drawing of a chained segment begins or ends (see GpiOpenSegment and GpiCloseSegment for more details).
- When an attribute-setting function (for example, GpiSetAttrs) that sets an attribute to its default value is issued, or interpreted in a retained segment during a drawing operation.

Each attribute has an initial default value, established when the presentation space is first created. The value of this is given under the appropriate GpiSet... call. The default value can be changed by GpiSetDefAttrs. Changing the default value takes effect immediately for the current value, if this is set to default at the time.

Each attribute of the primitive type in question is represented by one flag in the **AttrMask** parameter. Any attribute for which the appropriate flag is set has its default value updated to the value specified in the **Attrs** structure. The default value of any attribute for which the appropriate flag in **AttrMask** is not set is unchanged.

The data in the **Attrs** buffer consists of a structure of attribute data. The layout of the structure is fixed for each primitive type. Only data for attributes for which the flag is set in **AttrMask** is inspected; any other data is ignored.

Programming Note: The buffer need be no longer than is necessary to contain the data for the highest offset attribute referenced.

If an attempt is made to set an invalid default value by this call, none of the specified default attribute values is changed. Note, however, that some invalid default attribute values (for example, certain color and mix values) may not be detected until the attribute is set to default and used, at which point the implementation optionally defaults them, or causes an error to be logged.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1. Also, in a metafile, the default line width (see GpiSetLineWidth) is always rounded to an integer value, as is the default character box (see GpiSetCharBox) for GPIF_SHORT format presentation spaces (see GpiCreatePS).

GpiSetDefaultViewMatrix — Set Default View Matrix

This call sets the default viewing transform that is to apply to the whole picture.

GpiSetDefaultViewMatrix (hps, Count, array, Options, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of elements.

The number of elements supplied in **array**, that are to be examined, starting from the beginning of the structure. If **Count** is less than 9, remaining elements default to the corresponding elements of the identity matrix. Specifying **Count** = 0 means that the identity matrix is used.

array *(MATRIX)* — input
Transformation matrix.

The elements of the transform, in row order. The first, second, fourth, and fifth elements are of type FIXED, and have an assumed binary point between the second and third bytes. Thus a value of 1.0 is represented by 65 536. Other elements are normal signed integers. If the presentation-space coordinate format is GPIF_SHORT (see GpiCreatePS), these elements must be within the range −1 through +1.

The third, sixth, and ninth elements, when specified, must be 0, 0, and 1, respectively.

Options *(LONG)* — input
Transform options.

Specifies how the transform defined by the **array** parameter should be used to modify the existing default viewing transform.

Possible values are:

TRANSFORM_REPLACE	The previous default viewing transform is discarded and replaced by the specified transform.
TRANSFORM_ADD	The specified transform is combined with the existing default viewing transform, in the order (1) existing transform, (2) new transform. This option is most useful for incremental updates to transforms.
TRANSFORM_PREEMPT	The specified transform is combined with the existing default viewing transform, in the order (1) new transform, (2) existing transform.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_TRANSFORM_TYPE
PMERR_INV_MATRIX_ELEMENT

GpiSetDefaultViewMatrix — Set Default View Matrix

SAA

Remarks

The transform matrix specified is used to update any previous default viewing transform, depending upon the value of **Options**.

The transform is specified as a one-dimensional array of **Count** elements, being the first n elements of a 3-row by 3-column matrix ordered by rows. The order of the elements is:

```
Matrix                  Array

⎡ a  b  0 ⎤
⎢ c  d  0 ⎥          (a,b,0,c,d,0,e,f,1)
⎣ e  f  1 ⎦
```

The transform acts on the coordinates of the primitives in a segment, so that a point with coordinates (x,y) is transformed to the point:

(a*x + c*y + e, b*x + d*y + f)

The initial value of the default viewing 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 (which only applies if the presentation space coordinate format, as set by the GpiCreatePS call, is GPIF_LONG), it is possible for the combined effect of this and any other relevant transforms to exceed fixed-point implementation limits. This causes an error.

This call must not be issued in a path or area bracket.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiSetDefTag – Set Default Tag

This call specifies the default value of the primitive tag (see GpiSetTag).

GpiSetDefTag (hps, Tag, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Tag *(LONG)* – input
Default tag identifier.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION

Remarks

The primitive tag is reset to its default value at the following times:

- When the presentation space is associated with a device context (see GpiAssociate).
- When GpiResetPS is issued.
- When drawing of a chained segment begins or ends (see GpiOpenSegment and GpiCloseSegment for more details).

The initial default value of the primitive tag, when the presentation space is first created, is 0. The default value can be changed by GpiSetDefTag. Changing the default value has an immediately effect on the current primitive tag, if this is currently set to the default value.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiSetDefViewingLimits – Set Default Viewing Limits

This call specifies the default value of the viewing limits (see GpiSetViewingLimits).

GpiSetDefViewingLimits (hps, Limits, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Limits *(RECT)* – input
Default viewing limits.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

The viewing limits are reset to their default value at the following times:

- When the presentation space is associated with a device context (see GpiAssociate).
- When GpiResetPS is issued.
- When drawing of a chained segment begins or ends (see GpiOpenSegment and GpiCloseSegment for more details).

The initial default value of the viewing limits, when the presentation space is first created, is no clipping. The default value can be changed by GpiSetDefViewingLimits. Changing the default values has an immediately effect on the current viewing limits, if these are currently set to the default value.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiSetDrawControl – Set Draw Control

This call sets various options for subsequent GpiDrawChain, GpiDrawFrom, GpiDrawSegment, and GpiDrawDynamics operations.

GpiSetDrawControl (hps, Control, Value, *Success***)**

Parameters

hps *(HPS)* – input
Presentation-space handle.

Control *(LONG)* – input
Drawing control:

DCTL_ERASE Erase before draw. Perform an implicit GpiErase operation before GpiDrawChain, GpiDrawFrom, or GpiDrawSegment. The output display area of the Device Context associated with the specified presentation space is cleared before drawing.

DCTL_DISPLAY * Enable drawing to occur on the output medium. If this control is set to off, then except for GpiErase, no output operations appear on the output medium. This includes raster operations, such as drawing primitives, and GpiDraw... operations.

DCTL_BOUNDARY * Accumulate boundary data. During any output operations except GpiErase, accumulate the bounding rectangle of the drawing.

DCTL_DYNAMIC Draw dynamic segments. Perform an implicit GpiRemoveDynamics before GpiDrawChain, GpiDrawFrom, or GpiDrawSegment, and an implicit GpiDrawDynamics afterwards.

Note that, to either remove or draw dynamic segments, the system forces the foreground mix to FM_XOR, and the background mix to BM_LEAVEALONE. If the first nondynamic segment being drawn (immediately after the dynamic segments have been removed) has the ATTR_FASTCHAIN attribute (see GpiSetInitialSegmentAttrs), it may be necessary for it to set the mix modes itself before drawing. Similar considerations may apply for any subsequent drawing after the dynamic segments have been replaced.

DCTL_CORRELATE * If this control is set, any GpiPutData, GpiElement, GpiPlayMetaFile, or individual drawing primitives which are passed across the API outside a segment bracket cause a correlation operation to be performed, and a return code to be set if a hit occurs. (Correlation inside segments, both retained and nonretained, is controlled by the segment attribute ATTR_DETECTABLE).

This control only has an effect in draw or draw-and-retain modes (see GpiSetDrawingMode).

Programming Note: Controls identified by an asterisk (*) above are the only ones relevant to a micro-presentation space. Any other control settings are ignored for a micro-presentation space.

Value *(LONG)* – input
Required value of the drawing control:

DCTL_OFF Set control off
DCTL_ON Set control on.

GpiSetDrawControl — Set Draw Control

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_DRAW_CONTROL
PMERR_INV_DRAW_VALUE
PMERR_INV_IN_SEG
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_IN_ELEMENT
PMERR_INV_MICROPS_DRAW_CONTROL

Remarks

The default values are DCTL_OFF for all controls other than DCTL_DISPLAY which is DCTL_ON.

This call must not be issued in any of these cases:

- Inside an open segment.
- Outside an open segment, but inside one of:
  - Area bracket
  - Element bracket
  - Path bracket.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles (see "Metafile Restrictions" on page D-1).

GpiSetDrawingMode — Set Drawing Mode

This call sets the drawing mode to control the handling of subsequent individual drawing primitive and attribute calls.

GpiSetDrawingMode (hps, Mode, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Mode *(LONG)* — input
Mode to be used for subsequent drawing calls:

DM_DRAW	Draw, unless in an unchained segment
DM_RETAIN	Retain, if within a segment
DM_DRAWANDRETAIN	Draw-and-retain, combination of above.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_IN_ELEMENT
PMERR_INV_IN_SEG
PMERR_INV_DRAWING_MODE

Remarks

The drawing mode affects the handling of subsequent individual drawing primitive and attribute calls, and the GpiPutData, GpiElement, and GpiPlayMetaFile calls.

Primitives and attributes can be drawn immediately, retained, or both, in the current segment.

Programming Note: Any primitive and attribute setting calls that occur **outside** a segment (that is, outside a GpiOpenSegment — GpiCloseSegment bracket) are always treated as nonretained. Conversely, any segments that are not chained are always retained. This table summarizes how the actual drawing mode is arrived at:

GpiSetDrawingMode Parameter	Context		
	Chained Segment	Unchained Segment	Outside Segment
DM_DRAWANDRETAIN	draw-and-retain	retain	draw
DM_RETAIN	retain	retain	draw
DM_DRAW	draw	retain	draw

The actual drawing mode (referred to when describing other Gpi calls) therefore depends upon the mode as set by GpiSetDrawingMode, together with the context, as in the table. It is this actual drawing mode that determines whether a drawing call is retained (**retain** or **draw-and-retain**), and whether it is drawn immediately (**draw** or **draw-and-retain**).

It is an error to try to set the drawing mode within a segment bracket, and also outside a segment bracket, if in one of the following:

- Area bracket
- Element bracket
- Path bracket.

The default drawing mode is DM_DRAW.

GpiSetEditMode — Set Edit Mode

This call sets the current editing mode.

GpiSetEditMode (hps, Mode, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Mode *(LONG)* — input
Edit mode:

SEGEM_INSERT	Insert mode
SEGEM_REPLACE	Replace mode.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_EDIT_MODE
PMERR_INV_IN_ELEMENT
PMERR_INV_MICROPS_FUNCTION

Remarks

This call determines whether a new element is to be inserted into a segment, moving any subsequent elements further along the segment, or whether a new element is to replace the current element.

In SEGEM_INSERT mode, when an element is generated, it is inserted following the element indicated by the element pointer. The element pointer is updated to point to the new element.

In SEGEM_REPLACE mode, when an element is generated, it replaces the element indicated by the element pointer. The element pointer does not change. It is an error if a new element is generated in SEGEM_REPLACE mode if the element pointer is zero (as it is when a segment is opened).

The editing mode can be changed at any time, (except while within an element bracket), and is not an attribute of a specific segment. It only applies to the storing of data within retained segments. It is not an error to issue this call in other drawing modes; the value of the edit mode is set irrespective of the value of the draw mode.

This call is invalid within an element bracket. The default editing mode (set by GpiCreatePS or GpiResetPS) is SEGEM_INSERT.

GpiSetElementPointer — Set Element Pointer

SAA

This call sets the element pointer, within the current segment, to the element number specified.

GpiSetElementPointer (hps, Element, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Element *(LONG)* — input
The element number required.

If the value specified is negative, the element pointer is set to zero.

If the value specified is greater than the number of elements in the segment, the element pointer is set to the last element.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_NOT_IN_RETAIN_MODE
PMERR_NO_CURRENT_SEG
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_IN_ELEMENT

Remarks

The currently open segment has an element pointer that points to a particular element in the segment; each element is placed into the segment at the place indicated by the pointer. When a retained 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 segment. When a segment is reopened, the element pointer is reset to zero (that is, before the first element).

The element pointer for a segment is not remembered if the segment is closed and subsequently reopened.

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress. It is invalid within an element bracket.

GpiSetElementPointerAtLabel — Set Element Pointer At Label

This call sets the element pointer, within the current segment, to the element containing the specified label.

GpiSetElementPointerAtLabel (hps, Label, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Label *(LONG)* — input
Required label.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_NOT_IN_RETAIN_MODE
PMERR_NO_CURRENT_SEG
PMERR_INV_IN_ELEMENT
PMERR_LABEL_NOT_FOUND

Remarks

The search starts from the element pointed to by the current element pointer. If the specified label is not found between there and the end of the segment, an error is generated and the element pointer is left unchanged (also see GpiSetElementPointer).

This call is only valid when the drawing mode (see GpiSetDrawingMode) is set to **retain** (not **draw-and-retain**), and a segment bracket is currently in progress. It is invalid within an element bracket.

GpiSetGraphicsField — Set Graphics Field

SAA

This call sets the size and position of the graphics field in presentation page units.

GpiSetGraphicsField (hps, Field, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Field *(RECT)* — input
Graphics field.

It is an error if the top coordinate is less than the bottom, or the right coordinate is less than the left.

All values are in presentation-page units.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_GRAPHICS_FIELD
PMERR_INV_COORDINATE

Remarks

The graphics field specifies a clipping boundary within the presentation page.

The boundaries are inclusive, so that points on them are not clipped (removed). By default, no clipping is performed.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiSetInitialSegmentAttrs — Set Initial Segment Attributes

This call specifies a segment attribute that is used when a segment is subsequently created.

GpiSetInitialSegmentAttrs (hps, Attribute, Value, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Attribute *(LONG)* — input
Segment attribute:

ATTR_DETECTABLE Detectability.

This can be used to determine whether a correlation function can be performed on the primitives within the segment. For correlation on retained segments see:

- GpiCorrelateChain
- GpiCorrelateFrom
- GpiCorrelateSegment.

Correlation on primitives outside segments is controlled by the correlate flag on draw controls (see GpiSetDrawControl).

ATTR_VISIBLE Visibility.

Controls whether a segment is to be made visible on the output medium.

ATTR_CHAINED Chained.

Controls whether the segment is a root segment to be included in the segment drawing chain. In **draw** or **draw-and-retain** modes (see GpiSetDrawingMode) a chained segment is drawn as it passes across the API; an unchained segment is not.

Unchained segments are usually called from another segment. They can also be segments that are inserted into the chain later (with GpiSetSegmentPriority or GpiSetSegmentAttrs), or segments that are drawn individually with GpiDrawSegment.

ATTR_DYNAMIC Dynamic.

Controls whether the segment is to be dynamic; that is, drawn using exclusive-OR, so that it can be readily erased by redrawing it. (See GpiDrawDynamics, GpiRemoveDynamics, and the DCTL_DYNAMIC option of GpiSetDrawControl.)

Only retained segments can be dynamic.

The dynamic segment attribute is always ignored if the segment is not currently chained.

ATTR_FASTCHAIN Fast chaining.

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

ATTR_PROP_DETECTABLE Propagate detectability.

Controls whether the value of the detectability attribute for a segment should be propagated (forced) to all segments beneath it in the hierarchy.

ATTR_PROP_VISIBLE Propagate visibility.

Controls whether the value of the visibility attribute for a segment should be propagated (forced) to all segments beneath it in the hierarchy.

Value *(LONG)* — input
Attribute value:

ATTR_ON On/yes
ATTR_OFF Off/no.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_ATTR
PMERR_INV_SEG_ATTR_VALUE
PMERR_INV_MICROPS_FUNCTION

Remarks

Initial segment attributes are modal settings used to determine the initial attributes of new segments as they are created; that is, when an GpiOpenSegment call is issued, and the segment does not already exist. The default values of initial segment attributes are:

- Not detectable
- Visible
- Chained
- Not dynamic
- Fast chaining
- Propagate detectability
- Propagate visibility.

A nonretained segment can never be given the **dynamic** attribute.

Primitives outside segments are not affected by GpiSetInitialSegmentAttrs.

GpiSetLineEnd — Set Line End

This call sets the current line-end attribute.

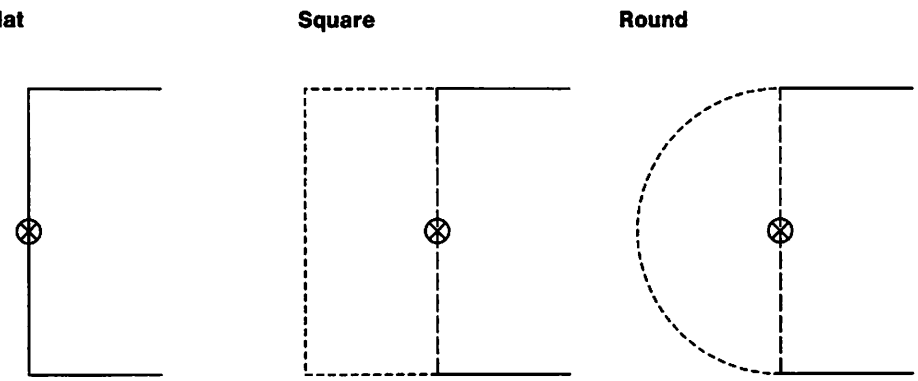
```
GpiSetLineEnd (hps, LineEnd, Success)
```

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineEnd *(LONG)* — input
Style of line end:

LINEEND_DEFAULT	Use default, same as LINEEND_FLAT (unless changed with GpiSetDefAttrs)
LINEEND_FLAT	Flat
LINEEND_SQUARE	Square
LINEEND_ROUND	Round.



⊗ Geometric point of line end

----- Outline of end shape

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LINE_END_ATTR

Remarks

This attribute defines the shape of the ends of lines or arcs at the beginning and end of an open figure. It is only used during a GpiModifyPath call, with a **Mode** parameter of MPATH_STROKE or during a GpiStrokePath call.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the line-end attribute is preserved.

GpiSetLineEnd — Set Line End

SAA

Graphic Elements and Orders

Element Type: OCODE_GSLE

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Line End

Element Type: OCODE_GPSLE

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Line End

GpiSetLineJoin — Set Line Join

This call sets the current line-join attribute.

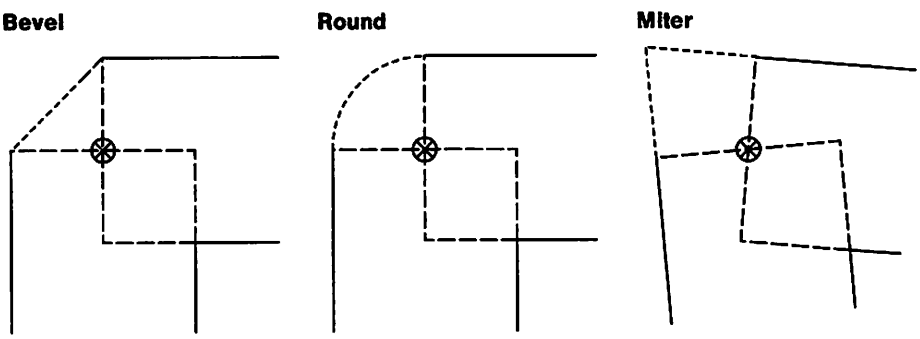
GpiSetLineJoin (hps, LineJoin, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineJoin *(LONG)* — input
Style of line join:

LINEJOIN_DEFAULT	Use default, same as LINEJOIN_BEVEL (unless changed with GpiSetDefAttrs)
LINEJOIN_BEVEL	Bevel
LINEJOIN_ROUND	Round
LINEJOIN_MITRE	Miter.



⊗ Geometric point of line join

----- Outline of join shape

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LINE_JOIN_ATTR

Remarks

This attribute defines how individual lines and arcs within a figure are joined together. It is only used during a GpiModifyPath call, with a **Mode** parameter of MPATH_STROKE, or during a GpiStrokePath call.

For LINEJOIN_MITRE, where the lines going into a join are nearly parallel (a very sharp change in direction), a miter join could potentially extend to a distance that approaches infinity. To prevent this, whenever the ratio of the miter length to the geometric line width exceeds **10**, a bevel join is drawn instead. (The miter length is the distance from the point at which the inner edges of the wideline intersect, to the point at which the outer edges of the wideline intersect.)

GpiSetLineJoin — Set Line Join

The attribute mode (see GpiSetAttrMode) determines whether the current value of the line-join attribute is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSLJ

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Line Join

Element Type: OCODE_GPSLJ

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Line Join

GpiSetLineType — Set Line Type

This call sets the current cosmetic line-type attribute.

GpiSetLineType (hps, LineType, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineType *(LONG)* — input
Line types available:

LINETYPE_DEFAULT	- Solid line (the default)	———————
LINETYPE_DOT	- Dotted line	.....
LINETYPE_SHORTDASH	- Short-dashed line	- - - - - - - - - -
LINETYPE_DASHDOT	- Dash-dot line	- · — · — · — · — · — · —
LINETYPE_DOUBLEDOT	- Double-dotted line	·· ·· ·· ·· ·· ·· ·· ·· ··
LINETYPE_LONGDASH	- Long-dashed line	— — — — — — — — — —
LINETYPE_DASHDOUBLEDOT	- Dash-double-dot line	· — ·· — ·· — ·· — ·· —
LINETYPE_SOLID	- Solid line	———————
LINETYPE_ALTERNATE	- Alternate pels on	.....
LINETYPE_INVISIBLE	- Invisible line	

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LINE_TYPE_ATTR

Remarks

A nonsolid line type consists of a sequence of "on" and "off" runs of pels that gives the appearance of a dotted or a dashed line, for example.

This attribute specifies the cosmetic line type, which is used for all line and curve drawing. It does not depend upon transforms, so that, for example, dashes do not become longer when a "zoom in" occurs.

The standard line types are implemented on each device to give a good appearance on that device, taking into account the pel resolution. Their definitions cannot be changed by applications, nor may applications define additional cosmetic line types.

GpiSetLineType — Set Line Type

The system maintains position within the line-type definition so that, for example, a curve may be implemented as a polyline. However, some functions cause position to be reset to the start of the definition. These are:

- GpiCallSegmentMatrix
- GpiMove
- GpiPop (or end of called segment) that pops current position or a model transform
- GpiSetCurrentPosition
- GpiSetLineType
- GpiSetModelTransformMatrix
- GpiSetPageViewport
- GpiSetSegmentTransformMatrix.

The default line-type is solid. This can be changed with GpiSetDefAttrs.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the line-type attribute is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSLT

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Line Type

Element Type: OCODE_GPSLT

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Line Type

GpiSetLineWidth — Set Line Width

This call sets the current cosmetic line-width attribute.

GpiSetLineWidth (hps, LineWidth, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineWidth *(ROF)* — input
Line-width multiplier:

LINEWIDTH_DEFAULT	Use default; same as LINEWIDTH_NORMAL (unless changed with GpiSetDefAttrs).
LINEWIDTH_NORMAL	Normal width (1.0).
	Any other positive value is a multiplier on the "normal" line width. Only "normal" line widths are supported; anything greater results in a warning.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LINE_WIDTH_ATTR
PMERR_UNSUPPORTED_ATTR_VALUE

Remarks

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 when a "zoom-in" occurs.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the line-width attribute is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSFLW

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Fractional Line Width

Element Type: OCODE_GPSFLW

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Fractional Line Width

GpiSetLineWidthGeom — Set Line Width Geom

SAA

This call sets the current geometric line width attribute.

GpiSetLineWidthGeom (hps, LineWidth, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

LineWidth *(ROL)* — input
Geometric line width.

The geometric line width in world coordinates. It must not be negative.

A thickness of zero results in an area of zero width. Because filling includes the boundaries, this results in the thinnest possible lines and arcs, regardless of what transforms are in force.

The initial default value of geometric line-width is 1. This can be changed with GpiSetDefAttrs.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_GEOM_LINE_WIDTH_ATTR

Remarks

The geometric line width is specified in world-coordinate units, so that, for example, the thickness varies on a zoom operation.

This attribute is only used during a GpiModifyPath function with a **Mode** of MPATH_STROKE, or during a GpiStrokePath call. It specifies the width to be used in converting the lines and arcs, of which the path is composed, into wide lines and arcs. The resulting shape is treated like an area, in that the boundaries are considered to be part of its interior; this means that the actual width of the lines and arcs is one pel wider than the geometric line width transformed to device coordinates.

Normal line and curve drawing uses only the cosmetic line width (see GpiSetLineWidth).

This function must not be issued in an area or path bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the geometric line width is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSSLW

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Stroke Line Width

Element Type: OCODE_GPSSLW

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Stroke Line Width

GpiSetMarker — Set Marker

SAA

This call sets the value of the marker-symbol attribute.

GpiSetMarker (hps, Symbol, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Symbol *(LONG)* — input
Marker symbol.

The identity of the required marker symbol. 0 selects the default marker symbol, a value in the range 1 through 255 identifies a symbol in the current marker set. Valid values in the default marker set are shown below, these symbols are not necessarily available with other marker sets:

MARKSYM_DEFAULT	The default; same as **MARKSYM_CROSS**
MARKSYM_CROSS	×
MARKSYM_PLUS	+
MARKSYM_DIAMOND	◇
MARKSYM_SQUARE	□
MARKSYM_SIXPOINTSTAR	✳
MARKSYM_EIGHTPOINTSTAR	✳
MARKSYM_SOLIDDIAMOND	◆
MARKSYM_SOLIDSQUARE	■
MARKSYM_DOT	•
MARKSYM_SMALLCIRCLE	○
MARKSYM_BLANK	(blank)

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MARKER_SYMBOL_ATTR

Remarks

This call must not be issued in an area bracket.

The default marker-symbol is a cross. This can be changed with GpiSetDefAttrs.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the marker attribute is to be preserved.

Graphic Elements and Orders

Element Type: OCODE_GSMT

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Marker Symbol

Element Type: OCODE_GPSMT

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Marker Symbol

GpiSetMarkerBox — Set Marker Box

SAA

This call sets the current marker-box attribute.

GpiSetMarkerBox (hps, Size, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Size *(SIZEROF)* — input
Size of marker box.

The size is specified in world coordinates. The fractional part of the value should be zero.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

The value of the marker-box attribute affects the size of markers that are selected from a vector font only. The size of markers that are selected from an image font is not affected by this attribute.

For default markers, this attribute only has an effect if the device supports the scaling of default markers; that is, the CAPS_SCALED_DEFAULT_MARKERS parameter in the CAPS_ADDITIONAL_GRAPHICS element of the device capabilities array returned by the DevQueryCaps call is set to 1.

This call must not be issued in an area bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the marker-box attribute is preserved.

The initial default value of the marker box is the size returned by DevQueryCaps (CAPS_MARKER_WIDTH and CAPS_MARKER_HEIGHT), for the currently associated device, converted to presentation page space.

The default value can be changed with GpiSetDefAttrs.

Graphic Elements and Orders

Element Type: OCODE_GSMC

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Marker Cell

Element Type: OCODE_GPSMC

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Marker Cell

GpiSetMarkerSet — Set Marker Set

This call sets the current marker-set attribute.

GpiSetMarkerSet (hps, Set, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Set *(LONG)* — input
Marker-set local identifier.

The identity (lcid) of the required marker set:

LCID_DEFAULT	Default (can be set explicitly with GpiSetDefAttrs)
1 – 254	Identifies a logical font.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MARKER_SET_ATTR
PMERR_HUGE_FONTS_NOT_SUPPORTED

Remarks

This call must not be issued in an area bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the marker-set attribute is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSMS

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Marker Set

Element Type: OCODE_GPSMS

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Marker Set

GpiSetMetaFileBits — Set Metafile Bits

This call transfers metafile data from application storage into a memory metafile.

GpiSetMetaFileBits (hmf, Offset, Length, Buffer, *Success*)

Parameters

hmf *(HMF)* — input
Metafile-memory handle.

Offset *(LONG)* — input
Offset.

Offset, in bytes, into the metafile data from where the transfer must start. This is used when the metafile data is too long to fit into a single application buffer.

Length *(LONG)* — input
Length of the metafile data.

Buffer *(BUFFER)* — input
Metafile data buffer.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HMF
PMERR_INV_METAFILE_LENGTH
PMERR_INV_METAFILE_OFFSET
PMERR_METAFILE_IN_USE

Remarks

The application must ensure that the data is in the correct format. It should not have been changed since it was created by GpiQueryMetaFileBits.

GpiSetMix — Set Mix

SAA

This call sets the current foreground mix attribute for each individual primitive type.

GpiSetMix (hps, MixMode, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

MixMode *(LONG)* — input
Mix mode.

Defines the color-mixing mode.

Mixing other than FM_LEAVEALONE or FM_OVERPAINT is done on the physical color index. In general, this corresponds to the color index of the logical color table if an indexed color table has been realized. In other circumstances, the color that results from such a mix cannot be predicted. Nevertheless, if FM_XOR is supported, for example, drawing the same object twice with a foreground mix of FM_XOR and a background mix of BM_LEAVEALONE with no intervening drawing in other mix modes, causes the object to be erased cleanly.

The currently associated device supports any of the mixes specified as supported in DevQueryCaps (CAPS_FOREGROUND_MIX_SUPPORT). Any other valid mixes may be supported for some primitive types, but otherwise results in FM_OVERPAINT. An error is raised only if the value specified is not one of those listed below.

Programming Note: Mixes marked with an asterisk (*) are mandatory for all devices, except that FM_OR is only mandatory for devices capable of supporting it. FM_XOR is mandatory only on displays.

FM_DEFAULT	Use default, the same as FM_OVERPAINT, unless changed with GpiSetDefAttrs
FM_OR	Logical-OR (*)
FM_OVERPAINT	Overpaint (*)
FM_XOR	Logical-XOR (*)
FM_LEAVEALONE	Leave alone (invisible) (*)
FM_AND	Logical-AND
FM_SUBTRACT	(inverse source) AND destination
FM_MASKSRCNOT	Source AND (inverse destination)
FM_ZERO	All zeros
FM_NOTMERGESRC	Inverse (source OR destination)
FM_NOTXORSRC	Inverse (source XOR destination)
FM_INVERT	Inverse (destination)
FM_MERGESRCNOT	Source OR (inverse destination)
FM_NOTCOPYSRC	Inverse (source)
FM_MERGENOTSRC	(Inverse source) OR destination
FM_NOTMASKSRC	Inverse (source AND destination)
FM_ONE	All ones.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MIX_ATTR

Remarks

The current values for each primitive type are updated. The attribute mode (see GpiSetAttrMode) determines whether the current value of the mix attribute is preserved.

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

Graphic Elements and Orders

Element Type: OCODE_GSMX

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Mix

Element Type: OCODE_GPSMX

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Mix

GpiSetModelTransformMatrix — Set Model Transform Matrix

SAA

This call sets the model transform matrix for subsequent primitives.

GpiSetModelTransformMatrix (hps, Count, Array, Options, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of elements in matrix.

The number of elements of **Array** to be examined, starting from the beginning of the structure. If **Count** is less than **9**, remaining elements default to the corresponding elements of the identity matrix. If **Count** = **0**, the identity matrix is used.

Array *(MATRIX)* — input
Transformation matrix.

The elements of the transform, in row order. The first, second, fourth, and fifth elements are of type FIXED, and have an assumed binary point between the second and third bytes. Thus a value of 1.0 is represented by 65 536. Other elements are normal signed integers. If the presentation space coordinate format is GPIF_SHORT (see GpiCreatePS), these elements must be within the range −1 through +1.

The third, sixth, and ninth elements, when specified, must be 0, 0, and 1, respectively.

Options *(LONG)* — input
Transform options.

Specifies how the transform defined by the **Array** should be used to modify the existing current model transform (the existing transform is the concatenation, in the current call context, of the instance, segment and model transforms, from the root segment downwards). Possible values are:

TRANSFORM_REPLACE	The previous model transform is discarded and replaced by the specified transform.
TRANSFORM_ADD	The specified transform is combined with the existing model transform, in the order (1) existing transform, (2) new transform. This option is most useful for incremental updates to transforms.
TRANSFORM_PREEMPT	The specified transform is combined with the existing model transform, in the order (1) new transform, (2) existing transform.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_TRANSFORM_TYPE

Remarks

The matrix is used to update the previous current model transform, depending upon the value of **Options**.

The transform is specified as a one-dimensional array of **Count** elements, being the first n elements of a 3-row by 3-column matrix ordered by rows. The order of the elements is:

```
Matrix                  Array

┌        ┐
│ a  b  0 │
│ c  d  0 │          (a,b,0,c,d,0,e,f,1)
│ e  f  1 │
└        ┘
```

The transform acts on the coordinates of the primitives in a segment, so that a point with coordinates (x,y) is transformed to the point:

```
(a*x + c*y + e, b*x + d*y + f)
```

If scaling values greater than unity are given (which only applies if the presentation space coordinate format as set by the GpiCreatePS call is GPIF_LONG) it is possible for the combined effect of this and any other relevant transforms to exceed fixed-point implementation limits. This causes an error.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the model transform is preserved.

Model transforms can apply to primitives either inside or outside segments.

Graphic Elements and Orders

Element Type: OCODE_GSTM

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Model Transform

Element Type: OCODE_GPSTM

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Model Transform

GpiSetPageViewport — Set Page Viewport

SAA

This call sets the page viewport within device space.

GpiSetPageViewport (hps, Viewport, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Viewport *(RECT)* — input
Page viewport.

The page viewport is specified in device units.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PAGE_VIEWPORT
PMERR_INV_COORDINATE

Remarks

The presentation page maps to the page viewport and together they define the device transform.

When a presentation space is associated with a device context, a default page viewport is set up.

The origin in device space is mapped to the bottom-left of the output media (window or paper, for example).

This call must not be issued when there is no device context associated with the presentation space.

GpiSetPattern — Set Pattern

This call sets the current value of the pattern symbol attribute.

GpiSetPattern (hps, PatternSymbol, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

PatternSymbol *(LONG)* — input
Pattern symbol.

Identifies the shading pattern to be used to fill areas. The actual pattern that appears depends on the particular pattern set selected by the pattern-set attribute. A value of 0 selects the default pattern, and values in the range 1 through 255 select particular patterns within the set.

Possible values if the default pattern set has been selected are:

Symbolic name	Description	Pattern number: see Figure 4-10
PATSYM_DEFAULT	The default; same as 16 (unless changed with GpiSetDefAttrs).	
PATSYM_DENSE1 through PATSYM_DENSE8	Solid shading with decreasing density	1 through 8
PATSYM_VERT	Vertical pattern	9
PATSYM_HORIZ	Horizontal pattern	10
PATSYM_DIAG1	Diagonal pattern 1, bottom left to top right	11
PATSYM_DIAG2	Diagonal pattern 2, bottom left to top right	12
PATSYM_DIAG3	Diagonal pattern 3, top left to bottom right	13
PATSYM_DIAG4	Diagonal pattern 4, top left to bottom right	14
PATSYM_NOSHADE	No shading	15
PATSYM_SOLID	Solid shading	16
PATSYM_HALFTONE	Alternate pels on	
PATSYM_BLANK	Blank (same as PATSYM_NOSHADE)	

Notes:

1. The pattern PATSYM_HALFTONE can be the same as PATSYM_DENSE4. On non-bit-mapped devices it may be mapped to another base pattern.
2. If the specified pattern is not valid, the default (device-dependent) pattern is used.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PATTERN_ATTR

Remarks

Any symbol from a raster font can be used as a pattern by the appropriate use of this call and the GpiSetPatternSet call.

If the current pattern set specifies a bit map (see GpiSetBitmapId and GpiSetPatternSet), the pattern attribute is ignored.

If **PatternSymbol** is set or defaulted to PATSYM_SOLID, and the **Set** parameter of GpiSetPatternSet is LCID_DEFAULT, pattern colors that are not available may be approximated by dithering (unless dithering has been disabled by setting the LCOL_PURECOLOR bit on the **Options** parameter of GpiCreateLogColorTable).

This call must not be issued in an area or path bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the pattern symbol is preserved.

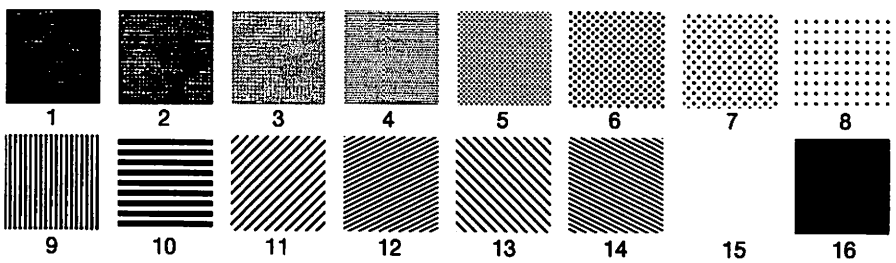


Figure 4-10. Shading patterns in the default pattern set

Graphic Elements and Orders

Element Type: OCODE_GSPT

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Pattern Symbol

Element Type: OCODE_GPSPT

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Pattern Symbol

GpiSetPatternRefPoint — Set Pattern Reference Point

This call sets the current pattern reference point to the specified value.

GpiSetPatternRefPoint (hps, RefPoint, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

RefPoint *(POINT)* — input
Pattern reference point.

The coordinates are world coordinates.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

The pattern reference point is the point to which the origin of the area filling pattern maps. The pattern is mapped into the area to be filled by conceptually replicating the pattern definition in a horizontal and vertical direction.

Because 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 also appears to move, so as to retain its position relative to the area boundaries.

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

This call must not be issued in an area or path bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the pattern reference point is preserved.

The initial default pattern reference point is (0,0). This can be changed with GpiSetDefAttrs.

GpiSetPatternRefPoint — Set Pattern Reference Point

SAA

Graphic Elements and Orders

Element Type: OCODE_GSPRP

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Pattern Reference Point

Element Type: OCODE_GPSPRP

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Pattern Reference Point

GpiSetPatternSet — Set Pattern Set

This call sets the current pattern-set attribute to the specified value.

GpiSetPatternSet (hps, Set, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Set *(LONG)* — input
Pattern-set local identifier:

LCID_DEFAULT	Default (can be set explicitly with GpiSetDefAttrs).
1 — 254	Identifies a logical font or a bit map.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PATTERN_SET_FONT
PMERR_HUGE_FONTS_NOT_SUPPORTED

Remarks

The bit map, or character within the font selected, is used for shading. On some devices, a simplified form of the bit map, or character, is used. For example, only a subset such as the first 8 by 8 pels may be used; also on a monochrome device a color bit map is converted to monochrome.

Some fonts are not suitable, and an error is returned if an attempt is made to set them as the current pattern set. These include device fonts that cannot be used for shading, and any kind of raster font for a plotter device.

This call must not be issued in an area or path bracket.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the pattern-set attribute is preserved.

GpiSetPatternSet — Set Pattern Set

Graphic Elements and Orders

Element Type: OCODE_GSPS

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Pattern Set

Element Type: OCODE_GPSPS

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Pattern Set

GpiSetPel — Set Pel

This call sets a pel, at a position specified in world coordinates, using the current (line) color and mix.

GpiSetPel (hps, Point, *Hits*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Point *(POINT)* — input
Position in world coordinates.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

This call is subject to all the usual clipping (clip path, clip region, viewing limits, graphics field, visible region), and no error is returned if the point is subject to clipping.

This call is independent of drawing mode (see GpiSetDrawingMode); the effect always occurs immediately, and it is not retained even if the drawing mode is **draw-and-retain** or **retain**. (Its effect is, however, recorded in a metafile, but note that this is only successful if the metafile is replayed on a similar device, with **draw** drawing mode.)

Programming Note: This call must not be used when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

GpiSetPickAperturePosition — Set Pick-Aperture Position

This call sets the center of the pick aperture, in presentation page space, for subsequent nonretained correlation operations.

GpiSetPickAperturePosition (hps, Pick, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Pick *(POINT)* — input
Center of the pick aperture.

The center is in presentation page coordinates.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

GpiSetPickApertureSize — Set Pick-Aperture Size

This call sets the pick-aperture size.

GpiSetPickApertureSize (hps, Options, Size, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Options *(LONG)* — input
Setting option:

PICKAP_DEFAULT	Use the default pick aperture. The value of **Size** is ignored.
PICKAP_REC	Use the values specified by **Size**.

Size *(SIZEROL)* — input
Pick aperture size.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PICK_APERTURE_OPTION
PMERR_INV_PICK_APERTURE_SIZE

Remarks

The pick aperture can be set either to the default value, or to a specified size in presentation page space. This is used in any subsequent nonretained or retained correlation operations.

The default size is a rectangle in presentation page space that produces a square on the device, with sides equal to the default character cell height.

GpiSetPS –
Set Presentation Space

This call sets the presentation space size, units, and format.

GpiSetPS (hps, size, Options, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

size *(SIZEROL)* – input
Presentation-space size.

Options – input
Options.

This contains fields of option bits. For each field, one value should be selected (unless the default is suitable). These values can then be ORed together to generate the parameter.

PS_UNITS *(BIT6)*
Presentation page size units.

Indicates the units for the presentation page size. In each case, the origin is at the bottom left. Possible values are:

PU_ARBITRARY	Application-convenient units
PU_PELS	Pel coordinates
PU_LOMETRIC	Units of 0.1 mm
PU_HIMETRIC	Units of 0.01 mm
PU_LOENGLISH	Units of 0.01 inch
PU_HIENGLISH	Units of 0.001 inch
PU_TWIPS	Units of 1/1440 inch.

PS_FORMAT *(BIT4)*
Coordinate format.

Indicates options to be used when storing coordinate values internally in the segment store.

For most calls, the format is not directly visible to an application. However, it is visible during editing (for example, GpiQueryElement). The format also has an effect on the amount of storage required for segment store.

One of these can be selected, for a GPIT_NORMAL presentation space (for a GPIT_MICRO presentation space, only GPIF_DEFAULT is allowed):

GPIF_DEFAULT	Default local format (same as GPIF_LONG)
GPIF_SHORT	2-byte integers
GPIF_LONG	4-byte integers.

PS_TYPE *(BIT1)*
Presentation space.

This option is ignored.

PS_MODE *(BIT1)*
Mode.

This option is ignored.

PS_ASSOCIATE *(BIT1)*
Association indicator.

This option is ignored.

PS_NORESET *(BIT1)*
Inhibit full reset indicator.

Inhibits the full reset of the presentation space. If this flag is set, a reset equivalent to GRES_SEGMENTS is performed. If it is not set, a full reset (GRES_ALL) is performed.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_HDC
PMERR_INV_PS_SIZE
PMERR_INV_OR_INCOMPAT_OPTIONS
PMERR_INV_FOR_THIS_DC_TYPE

Remarks

The presentation space is re-initialized to the same state that occurs as if it had been created using the specified size and option values. However, whether the presentation space is a micro presentation space or a normal presentation space cannot be changed, and any device context which is already associated remains associated.

The presentation space code page is set to the current process code page.

On completion, the presentation space is reset with the equivalent of GRES_ALL (see GpiResetPS), unless **PS_NORESET** is specified, in which case only the equivalent of a GRES_SEGMENTS reset is performed.

Note: This call cannot be used to a device context of type OD_METAFILE, OD_METAFILE_NOQUERY, or OD_QUEUED.

GpiSetRegion — Set Region

SAA

This call changes a region to be the logical-OR of a set of rectangles.

GpiSetRegion (hps, hrgn, count, Rectangles, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

The region must be owned by the device identified by the currently associated device context.

hrgn *(HRGN)* — input
Region handle.

count *(LONG)* — input
Count of rectangles.

This is the number of rectangles specified in **Rectangles**. If **count** = **0**, the region is set to EMPTY, and **Rectangles** is ignored.

Rectangles *(RECT****count)*** — input
Array of rectangles.

The rectangles are specified in device coordinates.

For each rectangle in the array, the value of **xright** must be greater than (or equal to) **xleft**, and **ytop** must be greater than (or equal to) **ybottom**.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_HRGN
PMERR_INV_COORDINATE
PMERR_INV_RECT
PMERR_REGION_IS_CLIP_REGION
PMERR_HRGN_BUSY

Remarks

This call is similar to GpiCreateRegion, except that it changes an already existing region to be the logical-OR of the supplied rectangles, instead of creating a new region.

The previous contents of the region are irrelevant. Points on the right-hand and top boundaries are not included in the changed region; points on the left-hand and bottom boundaries, that are not also on the right-hand or top boundaries, (that is, the top-left and bottom-right corner points) are included.

It is invalid if the specified region is currently selected as the clip region (by GpiSetClipRegion).

GpiSetSegmentAttrs — Set Segment Attributes

This call sets a segment attribute.

GpiSetSegmentAttrs (hps, SegId, Attribute, Value, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

SegId *(LONG)* — input
Segment identifier.

The identifier of the segment whose attribute is to be updated. It must be greater than zero.

Attribute *(LONG)* — input
Segment attribute.

For details of the following attributes, see the GpiSetInitialSegmentAttrs call.

ATTR_DETECTABLE	Detectability
ATTR_VISIBLE	Visibility
ATTR_CHAINED	Chained
ATTR_DYNAMIC	Dynamic
ATTR_FASTCHAIN	Fast chaining
ATTR_PROP_DETECTABLE	Propagate detectability
ATTR_PROP_VISIBLE	Propagate visibility.

Value *(LONG)* — input
Attribute value:

ATTR_ON	On/yes
ATTR_OFF	Off/no.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_SEG_ATTR
PMERR_INV_SEG_ATTR_VALUE
PMERR_SEG_NOT_FOUND
PMERR_INV_MICROPS_FUNCTION

GpiSetSegmentAttrs — Set Segment Attributes

Remarks

This call sets the value of one segment attribute for the specified segment. The segment can be any retained segment.

If the identifier is that of the currently-open segment:

- In **retain** mode, this is valid.
- In **draw-and-retain** mode, the retained segment is updated, but there is no change to the immediate drawing.
- In **draw** mode, it is invalid.

(For a description of drawing mode, see GpiSetDrawingMode).

When a segment is modified from nonchained to chained, it is added to the end of the drawing chain.

GpiSetSegmentPriority — Set Segment Priority

This call changes the position of a segment within the segment chain, or adds an unchained segment to the chain.

GpiSetSegmentPriority (hps, SegId, RefSegId, Order, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

SegId *(LONG)* — input
Segment identifier.

The identifier of the segment whose priority is to be changed; it must be greater than 0.

RefSegId *(LONG)* — input
Reference segment identifier.

The segment that identifies a position in the segment chain. The segment specified in the **SegId** parameter is placed either immediately before or after this segment, depending on the value specified in the **Order** parameter. Specifying 0 for **RefSegId** indicates that the position is to be the beginning or the end of the segment chain as defined by the value in the **Order** parameter.

Order *(LONG)* — input
Segment higher or lower.

Specifies whether the segment named in the **SegId** parameter is to be placed before or after the segment named in the **RefSegId** parameter. Possible values are:

LOWER_PRI The segment named in the **SegId** parameter is to have a lower priority than the segment named in the **RefSegId** parameter. The **SegId** segment is placed before the **RefSegId** segment. If 0 is specified in the **RefSegId** parameter, the segment identified in the **SegId** parameter is placed as the highest priority segment.

HIGHER_PRI The segment named in the **SegId** parameter is to have a higher priority than the segment named in the **RefSegId** parameter. The **SegId** segment is placed after the **RefSegId** segment. If 0 is specified in the **RefSegId** parameter, the segment identified in the **SegId** parameter is placed as the lowest priority segment.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_ORDERING_PARM
PMERR_SEG_AND_REFSEG_ARE_SAME
PMERR_SEG_NOT_FOUND
PMERR_INV_MICROPS_FUNCTION

GpiSetSegmentPriority — Set Segment Priority

SAA

Remarks

The specified segment can be a segment that exists in the segment chain, or an unchained segment. The effect of this call on an unchained segment is to add it to the segment chain in the specified position.

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 are placed on top of earlier ones (assuming a default mix mode) and 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.

GpiSetSegmentTransformMatrix — Set Segment Transform Matrix

This call sets the segment transform that normally applies to all of the primitives in the specified segment.

GpiSetSegmentTransformMatrix (hps, SegId, Count, array, Options, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

SegId *(LONG)* — input
Segment identifier.

This must be greater than 0.

Count *(LONG)* — input
Number of elements.

The number of elements to be used in the **array** parameter. If **Count** is less than 9, the elements omitted default to the corresponding elements of the identity matrix (see below). Specifying **Count**=**0** denotes that the identity matrix is used.

array *(MATRIX)* — input
Transformation matrix.

The elements of the transform, in row order. The first, second, fourth, and fifth elements are of type FIXED, and have an assumed binary point between the second and third bytes. Thus, a value of 1.0 is represented by 65 536. Other elements are normal signed integers. If the presentation space coordinate format is GPIF_SHORT (see GpiCreatePS), these elements must be within the range −1 through +1.

The third, sixth, and ninth elements, when specified, must be 0, 0, and 1, respectively.

Options *(LONG)* — input
Transform options.

Specifies how the existing segment transform is to be modified by the transform defined by the **array** parameter. The new segment transform is computed, and the result stored back in the segment, replacing the existing value. When the segment is drawn, the stored segment transform is used to update the segment transform that is currently in effect, in an additive manner. Possible values are:

TRANSFORM_REPLACE	The previous default segment transform is discarded and replaced by the specified transform.
TRANSFORM_ADD	The specified transform is combined with the existing default segment transform, in the order (1) existing transform, (2) new transform. This option is most useful for incremental updates to transforms.
TRANSFORM_PREEMPT	The specified transform is combined with the existing default segment transform, in the order (1) new transform, (2) existing transform.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

GpiSetSegmentTransformMatrix — Set Segment Transform Matrix

SAA

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_SEG_NAME
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_LENGTH_OR_COUNT
PMERR_SEG_NOT_FOUND
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_TRANSFORM_TYPE

Remarks

The matrix is used to update the segment transform of a retained segment, according to the value of the **Options** parameter.

The segment transform is actually a model transform that applies at the start of the segment. It can be overridden later in the segment with a GpiSetModelTransformMatrix call.

This call specifies the transform as a one-dimensional array of **Count** elements, being the first **Count** elements of a 3-row by 3-column matrix ordered in rows. The order of the elements is:

```
Matrix                  Array

┌       ┐
│ a  b  0 │
│ c  d  0 │          (a,b,0,c,d,0,e,f,1)
│ e  f  1 │
└       ┘
```

The transform acts on the coordinates of the primitives in a segment, so that a point with coordinates (x,y) is transformed to the point:

```
(a*x + c*y + e, b*x + d*y + f)
```

The initial value of the transform of a segment 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 (which only applies if the presentation space coordinate format, as set by the GpiCreatePS call, is GPIF_LONG) it is possible for the combined effect of this, and any other relevant transforms, to exceed fixed-point implementation limits. This causes an error.

Segment transforms do not apply to primitives outside segments.

GpiSetStopDraw — Set Stop Draw

This call sets or clears the "stop draw" condition.

GpiSetStopDraw (hps, Value, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Value *(LONG)* — input
Stop draw condition:

SDW_OFF Clear the "stop draw" condition
SDW_ON Set the "stop draw" condition.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_INV_STOP_DRAW_VALUE
PMERR_INV_MICROPS_FUNCTION

Remarks

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

The operations are:

- GpiDrawChain
- GpiDrawDynamics
- GpiDrawFrom
- GpiDrawSegment
- GpiPlayMetaFile
- GpiPutData.

The operation terminates with a warning.

This call 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 realizes 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 calls.

Any operation other than GpiSetStopDraw, directed at a presentation space that is currently in use, gives an error return code.

Note: If this call is issued when an asynchronous draw to a metafile is taking place, an unusable metafile results.

GpiSetTag – Set Tag

SAA

This call specifies a tag by which the following primitives are to be known.

GpiSetTag (hps, Tag, *Success*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Tag *(LONG)* – input
Tag identifier.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION

Remarks

When GpiCorrelateChain, GpiCorrelateFrom, or GpiCorrelateSegment is used to locate an object, both the segment identifier and the primitive tag of the object are returned to the application program.

If a tag of 0 is specified, the primitives have no name and are not returned by the correlate call.

Initially, the default and current tag are 0. The default tag can be changed with GpiSetDefTag.

Primitives within an unnamed segment cannot be picked or correlated, and any tag applied to them is ignored.

This call is not allowed between GpiBeginArea and GpiEndArea calls, therefore, all primitives within an area have the same tag.

The attribute mode (see GpiSetAttrMode) determines whether the current value of the tag is preserved.

Graphic Elements and Orders

Element Type: OCODE_GSPIK

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Pick Identifier

Element Type: OCODE_GPSPIK

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Pick Identifier

GpiSetViewingLimits — Set Viewing Limits

This call establishes a clipping rectangle in model space.

GpiSetViewingLimits (hps, Limits, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Limits *(RECT)* — input
Viewing limits in model space.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_COORDINATE

Remarks

Viewing limits can be set within a segment, and apply to all subsequent primitives in the segment and any segments it calls. They can be changed at any time within the segment and they are not subject to segment or model transformations. Limits specified in called segments override those set by the limits of the root segment.

The limits are reset to their default value at the start of each root segment, subject to the fast-chaining attribute, like primitive attributes. The initial default value is no clipping; this can be changed with GpiSetDefViewingLimits.

The boundaries are inclusive, so that points on them are not clipped (removed). If either the left boundary of **Limits** is greater than the right, or the bottom greater than the top, a NULL rectangle is defined. All points are clipped.

Attribute mode (see GpiSetAttrMode) has no effect on this call.

The viewing limits are converted under the current viewing and default viewing transformations to a clipping rectangle in the page. This remains in force until changed by a subsequent GpiSetViewingLimits call. Clipping actually takes place to the intersection of the viewing limits, the clip path, the clip region, the graphics field, and the client area on the device.

GpiSetViewingLimits — Set Viewing Limits

Graphic Elements and Orders

Element Type: OCODE_GSVW

This element type is generated if the attribute mode (see GpiSetAttrMode) is set to AM_NOPRESERVE.

Order: Set Viewing Window

Element Type: OCODE_GPSVW

This element type is generated if the attribute mode is set to AM_PRESERVE.

Order: Push and Set Viewing Window

GpiSetViewingTransformMatrix — Set Viewing Transform Matrix

This call sets the viewing transform that is to apply to any subsequently opened segments.

GpiSetViewingTransformMatrix (hps, Count, Array, Options, *Success***)**

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(LONG)* — input
Number of elements.

The number of elements supplied in **Array**, that are to be examined, starting from the beginning of the structure. If **Count** is less than **9**, remaining elements default to the corresponding elements of the identity matrix. Specifying **Count** = **0** means that the identity matrix is used.

Array *(MATRIX)* — input
Transformation matrix.

The elements of the transform, in row order. The first, second, fourth, and fifth elements are of type FIXED, and have an assumed binary point between the second and third bytes. Thus a value of 1.0 is represented by 65 536. Other elements are normal signed integers. If the presentation space coordinate format is GPIF_SHORT (see GpiCreatePS), these elements must be within the range −1 through +1.

The third, sixth, and ninth elements, when specified, must be 0, 0, and 1, respectively.

Options *(LONG)* — input
Transform option.

Specifies how the specified transform is to be used to modify the existing viewing transform. This must be:

TRANSFORM_REPLACE New/replace. The previous viewing transform is discarded and replaced by the specified transform.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MATRIX_ELEMENT
PMERR_INV_TRANSFORM_TYPE
PMERR_INV_IN_SEG
PMERR_NOT_IN_RETAIN_MODE

GpiSetViewingTransformMatrix — Set Viewing Transform Matrix

SAA

Remarks

This call is only valid outside segments. The viewing transform that is set applies to all subsequently opened (new) segments (it has no effect on primitives outside segments). All graphics primitives in a segment must have the same viewing transform. When it has been set for a particular segment, the viewing transform for that segment cannot be changed.

The transform is specified as a one-dimensional array of **Count** elements, being the first n elements of a 3-row by 3-column matrix ordered by rows. The order of the elements is:

```
Matrix                  Array

┌       ┐
│ a  b  0 │
│ c  d  0 │          (a,b,0,c,d,0,e,f,1)
│ e  f  1 │
└       ┘
```

The transform acts on the coordinates of the primitives in a segment, so that a point with coordinates (x,y) is transformed to the point:

```
(a*x + c*y + e, b*x + d*y + f)
```

The initial value of the viewing 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 │
└       ┘
```

The viewing transform must be set (or defaulted) to the unity transform, before any segment that is to be called is first opened.

If scaling values greater than unity are given (which only applies if the presentation space coordinate format, as set by the GpiCreatePS call, is GPIF_LONG) it is possible for the combined effect of this and any other relevant transforms to exceed fixed-point implementation limits. This causes an error.

This call must not be issued in a path or area bracket.

GpiStrokePath – Stroke Path

This call strokes a path, and then draws it.

GpiStrokePath (hps, Path, Options, *Hits*)

Parameters

hps *(HPS)* – input
Presentation-space handle.

Path *(LONG)* – input
Identifier of path to be stroked; it must be 1.

Options *(BIT32)* – input
Stroke option:

Reserved; must be zero.

Hits *(LONG)* – return
Correlation/error indicator:

GPI_OK	Successful
GPI_HITS	Correlate hit(s)
GPI_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_PATH_ID
PMERR_INV_RESERVED_FIELD
PMERR_PATH_UNKNOWN

Remarks

The path is first converted to one that describes the envelope of a wide line stroked using the current geometric line-width attribute (see GpiSetLineWidthGeom).

Programming Note: This call and GpiModifyPath are the only calls that can cause geometric wide lines to be constructed. For more details about the way in which the envelope is constructed, see GpiModifyPath.

The converted path is then filled, using winding mode area fill and the area attributes. The boundaries of the wide line are included in the fill.

When it has been drawn, the path is deleted.

This call is equivalent to GpiModifyPath, followed by GpiFillPath. It is provided to enable device drivers to optimize storage, if possible.

If the current drawing mode (see GpiSetDrawingMode) is **draw** or **draw-and-retain**, drawing occurs on the currently associated device. If the drawing mode is **retain**, this call is stored in the current segment and output occurs when the segment is subsequently drawn in the usual way.

GpiStrokePath – Stroke Path

Graphic Elements and Orders

Element Type: OCODE_GFPTH

Note that GpiFillPath also generates this element type.

Order: Fill Path

GpiTranslate — Translate Matrix

This call applies a translation to a transform matrix.

GpiTranslate (hps, Array, Options, Translation, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Array *(MATRIX)* — input/output
Transform matrix.

The elements of the transform, in row order. The first, second, fourth, and fifth elements are of type FIXED, and have an assumed binary point between the second and third bytes. Thus a value of 1.0 is represented by 65 536. Other elements are normal signed integers.

The third, sixth, and ninth elements must be 0, 0, and 1, respectively.

Options *(LONG)* — input
Transform options.

Specifies how the transform defined by the specified translation should be used to modify the previous transform specified by the **Array** parameter. Possible values are:

TRANSFORM_REPLACE	The previous transform is discarded and replaced by the transform describing the specified translation.
TRANSFORM_ADD	The previous transform is combined with a transform representing the specified translation in the order (1) previous transform, (2) translational transform. This option is most useful for incremental updates to transforms.

Translation *(POINT)* — input
Translation.

The coordinates of a point, relative to the origin, which defines the required translation.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_TRANSFORM_TYPE

GpiTranslate – Translate Matrix

Remarks

This call is a helper function which either applies a specified translational component to an existing transform matrix, or replaces the matrix with one which represents the specified translation alone.

The transform is specified as a one-dimensional array of 9 elements that are the elements of a 3-row by 3-column matrix ordered by rows. The order of the elements is:

```
Matrix                  Array

┌        ┐
│ a  b  0 │
│ c  d  0 │          (a,b,0,c,d,0,e,f,1)
│ e  f  1 │
└        ┘
```

Transforms act on the coordinates of primitives, so that a point with coordinates (x,y) is transformed to the point:

```
(a*x + c*y + e, b*x + d*y + f)
```

The transform can be used in any call following:

- GpiSetModelTransformMatrix
- GpiSetSegmentTransformMatrix
- GpiSetViewingTransformMatrix
- GpiSetDefaultViewMatrix.

Other similar helper functions are:

- GpiScale to apply a scaling component
- GpiRotate to apply a rotation component.

GpiUnloadFonts — Unload Fonts

This call unloads any font(s) previously loaded from the resource file by GpiLoadFonts.

GpiUnloadFonts (hab, Filename, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Filename *(STRL)* — input
Fully qualified file name of the font resource.

The file name extension is ".FON."

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_FONT_FILE_NOT_LOADED
PMERR_OWN_SET_ID_REFS
PMERR_OTHER_SET_ID_REFS (warning

Remarks

Before issuing this call, the application must:

1. Issue GpiSetCharSet to a font other than one of those to be unloaded, for example, to the default font.
2. Issue GpiDeleteSetId for each local identifier (lcid) that references one of the fonts (the LCID_ALL option can be used if all lcids are to be deleted).

An error is returned if lcids that reference one of the fonts still exist for this application, and a warning is logged if lcids exist for another application.

GpiUnrealizeColorTable — Unrealize Color Table

SAA

This call unrealizes the logical color table.

GpiUnrealizeColorTable (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY

Remarks

This call has the opposite effect of GpiRealizeColorTable. It causes the default device physical color table to be reinstated, and should be issued when the associated window ceases to be maximized.

The logical color table remains unchanged.

GpiWCBitBlt — World Coordinates Bit Blt

This call copies a rectangle of bit-map image data.

GpiWCBitBlt (Target, Source, Count, Points, Rop, Options, *Hits*)

Parameters

Target *(HPS)* — input
Target presentation-space handle.

Source *(HBITMAP)* — input
Source bit-map handle.

It is an error if this bit map is currently selected into a memory device context.

Count *(LONG)* — input
Point count.

This count must be equal to 4.

Points *(POINT*Count)* — input
Point array

Array of **Count** points, in the order **Tx1, Ty1, Tx2, Ty2, Sx1, Sy1, Sx2, Sy2**. These are:

Tx1,Ty1 Specify the bottom-left corner of the target rectangle in target world coordinates.
Tx2,Ty2 Specify the top-right corner of the target rectangle in target world coordinates.
Sx1,Sy1 Specify the bottom-left corner of the source rectangle in source device coordinates.
Sx2,Sy2 Specify the top-right corner of the source rectangle in source device coordinates.

Rop *(LONG)* — input
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 the 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, and T pel values, the final target value for the pel is determined by the appropriate **Rop** bit value as shown below:

P	S	T (initial)	T (final)
0	0	0	Index bit 0 (least significant)
0	0	1	Index bit 1
0	1	0	Index bit 2
0	1	1	Index bit 3
1	0	0	Index bit 4
1	0	1	Index bit 5
1	1	0	Index bit 6
1	1	1	Index bit 7 (most significant)

The index formed in the above way determines the mixing required. Mnemonic names are available for commonly used mixes:

```
ROP_SRCCOPY       /* SRC                     */
ROP_SRCPAINT      /* SRC OR DST              */
ROP_SRCAND        /* SRC AND DST             */
ROP_SRCINVERT     /* SRC XOR DST             */
ROP_SRCERASE      /* SRC AND NOT(DST)        */
ROP_NOTSRCCOPY    /* NOT(SRC)                */
ROP_NOTSRCERASE   /* NOT(SRC) AND NOT(DST)   */
ROP_MERGECOPY     /* SRC AND PAT             */
ROP_MERGEPAINT    /* NOT(SRC) OR DST         */
ROP_PATCOPY       /* PAT                     */
ROP_PATPAINT      /* NOT(SRC) OR PAT OR DST  */
ROP_PATINVERT     /* DST XOR PAT             */
ROP_DSTINVERT     /* NOT(DST)                */
ROP_ZERO          /* 0                       */
ROP_ONE           /* 1                       */
```

Options *(BIT32)* — input
Options.

How eliminated lines or columns are treated if a compression is performed.

Flags 15 through 31 of **Options** can be used for privately-supported modes for particular devices.

BBO_OR The default. If compression is necessary, logical-OR eliminated rows or columns. This is useful for white on black.

BBO_AND If compression is necessary, logical-AND eliminated rows or columns. This is useful for black on white.

BBO_IGNORE If compression is necessary, ignore eliminated rows or columns. This is useful for color.

Hits *(LONG)* — return
Correlation/error indicator:

GPI_OK Successful
GPI_HITS Correlate hit(s)
GPI_ERROR Error.

Possible returns from WinGetLastError:

PMERR_INV_HPS
PMERR_PS_BUSY
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_BITBLT_MIX
PMERR_INV_BITBLT_STYLE
PMERR_BITMAP_NOT_FOUND
PMERR_INV_COORDINATE
PMERR_INV_RECT
PMERR_NO_BITMAP_SELECTED
PMERR_INCORRECT_DC_TYPE
PMERR_INCOMPATIBLE_BITMAP
PMERR_INV_HBITMAP
PMERR_HBITMAP_BUSY

Remarks

A rectangle of bit-map image data is copied from a bit map, to a bit map selected into a device context associated with the target presentation space. Alternatively, the target presentation space can be associated with a device context that specifies a suitable raster device, for example, the screen.

Programming Note: In either case, both source and target device contexts must apply to the same physical device. It is an error if this device does not support raster operations.

A rectangle is specified in device coordinates for the source bit map, and one in world coordinates for the target presentation space. The source rectangle is noninclusive; the left and lower boundaries in device space are included, but not the right and upper boundaries. Thus if the bottom-left is equal to the top-right, the rectangle is deemed to be empty. The target rectangle is "inclusive-inclusive"; that is, all boundaries are included in the rectangle.

If the target rectangle, after transformation to device coordinates and adjustment for inclusivity, is not the same size as the source rectangle, then stretching or compressing of the data occurs. **Options** specifies how eliminated rows or columns of bits are to be treated if compression occurs. Note that the pattern data is never stretched or compressed.

Even if there is a rotational effect in the transformations, the data itself is not rotated.

The target rectangle is transformed to device coordinates, and if any shear or rotation has occurred, this is then converted to an upright rectangle that bounds the transformed figure. This rectangle is used as the target of the operation. No inversion of the image takes place.

These current attributes of the target presentation space are used (other than for converting between monochrome and color, as described below):

- Area color
- Area background color
- Pattern set
- Pattern symbol.

The color values are used in conversion between monochrome and color data. This is the only format conversion performed by this call. The conversions are:

- Output of a monochrome pattern to a color device.

  In this instance the pattern is converted first to a color pattern, using the current area colors:

  - source 1s → area foreground color
  - source 0s → area background color.

- Copying from a monochrome bit map to a color bit map (or device).

  The source bits are converted as follows:

  - source 1s → image foreground color
  - source 0s → image background color.

- Copying from a color bit map to a monochrome bit map (or device).

  The source bits are converted as follows:

  - source nonzeros → image foreground color
  - source 0s → image background color.

If the mix (**Rop**) does not call for a pattern, the pattern set and pattern symbol are not used.

Neither the source nor the pattern is required when a bit map, or part of a bit map, is to be cleared to a particular color.

If the mix does require both source and pattern, a three-way operation is performed.

If a pattern is required, dithering may be performed for solid patterns in a color that is not available on the device. See GpiSetPattern.

GpiWCBitBlt — World Coordinates Bit Blt

SAA

This call (unlike GpiBitBlt) can be drawn immediately, retained in segment store, or both of these, depending upon the drawing mode (see GpiSetDrawingMode).

Programming Note: There are restrictions on the use of this call when creating SAA-conforming metafiles; see "Metafile Restrictions" on page D-1.

Graphic Elements and Orders

Element Type: OCODE_GBBLT

Order: Bit Blt

Chapter 5. Picture Function Calls

The following table shows the Picture (PIC) function calls in alphabetic order according to their C/MASM name; the COBOL/FORTRAN name is also given.

C/MASM name	COBOL/FORTRAN name
Piclchg	PICONV
PicPrint	PIPRT

PicIchg –
Picture Interchange Convert

This call converts an interchange file in picture interchange format (PIF) into a metafile. It can also convert a font to a symbol set or a symbol set to a font.

```
PicIchg (hab, SourceFileName, DestinationFileName, Type, Success)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

SourceFileName *(STRL)* – input
The name of the source file.

DestinationFileName *(STRL)* – input
The name of the destination file.

It is an error if this file already exists.

Type *(LONG)* – input
Specifies the type of conversion.

PIC_PIFTOMF PIF to metafile (see remark 2)
PIC_SSTOFONT Symbol set to font (see remark 3 on page 5-3)

Success *(BOOL)* – return
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_CONV
PMERR_INV_SEGLEN
PMERR_DUP_SEGNAME
PMERR_INV_XFORM
PMERR_INV_VIEWLIM
PMERR_INV_3DCOORD
PMERR_SMB_OVFLOW
PMERR_SEG_OVFLOW

Remarks

1. EBCDIC Code page.

   The EBCDIC code page used for a PIF file and symbol set is derived from the country code defined by OS/2.

2. PIF to Metafile.

   Full PIF is supported.

   Restrictions:

   - Symbol and Pattern Sets.

     Any reference to an internal symbol or pattern set is changed to a reference to the default font character set (lcid 0).

   - Line Type Sets.

     Any reference to a line type set is changed to a reference to the default line type (line type 0).

3. Symbol Set to Font.

   Restrictions:

   - Only vector symbol sets (type 8) in SMB format are supported.

PicPrint — Picture Print

This call prints a metafile or picture interchange format (PIF) file.

```
PicPrint (hab, Filename, Type, Params, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Filename *(STRL)* — input
Full path and filename of the source file.

Type *(LONG)* — input
Type of source file:

PIP_MF Metafile
PIP_PIF PIF.

Params *(STRL)* — input

A string giving a list of parameters to be passed to the spooler and queue processor.

COP=n
where 'n' is the number of copies of this file to be printed. It must be in the range 1 through 999.

ARE=C | w,h,l,t
The ARE parameter determines the size and position of the output area.

A value of ARE=C means that the output area is the same size as the clip limits of the device.

To size and position the output at a specific point on the page, use ARE=w,h,l,t, where

w,h are the width and height of the desired output area

l,t are the offsets of the top-left corner of the output area from the left and top, respectively, of the maximum output area.

These four values must be given as percentages of the maximum output dimension.

The default is ARE=C.

FIT=S | l,t
The FIT parameter determines which part of the picture is to be printed. You can ask for the whole of the picture, scaled to fit the output area, or you can position the picture (real size) anywhere within the output area. This may mean that the picture is clipped at the boundaries of the output area.

A value of FIT=S causes the output to be scaled until the larger of the height or the width just fits within the defined output area. The aspect ratio of the picture is maintained.

To print the picture real size, use FIT=l,t, where l,t are the coordinates of the point in the picture that you want positioned at the center of the output area. l is measured from the left edge of the picture, and t is measured from the top edge. The coordinates must be given as percentages of the real dimensions of the picture.

The default is FIT=S.

COL=M | C The COL parameter allows you to specify color output if you have a color printer.

A value of COL=M creates monochrome output (black foreground with no background color). This is supported by all devices.

A value of COL=C creates colored output. If you request colored output on a monochrome device, the presentation driver tries to meet your request.

The default is COL=M.

MAP=N | A The MAP parameter allows you to decide how the neutral colors (those that are not specified in the picture file) are printed.

A value of MAP=N gives a normal representation of a screen picture on the printed page. This means that the page background is white and the foreground is black.

A value of MAP=A gives the reverse of the normal representation. This means that, on the printed page, the background is black and the foreground is white.

The default is MAP=N.

FORM=str The FORM parameter allows you to specify which form the output is printed on. The form must be valid for the printer.

Note: These parameters must be separated by one or more blank spaces, for example:

```
COP=3 ARE=C FIT=S
```

The parameters can be listed in any order. Errors in the list are ignored, and default values are used for parameters that are omitted or entered incorrectly.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_TYPE

Chapter 6. Profile Function Calls

The following table shows all the Profile (Prf) function calls in alphabetic order according to their C/MASM name; the COBOL/FORTRAN name is also given.

C/MASM name	COBOL/FORTRAN name
PrfAddProgram	PRADPR
PrfChangeProgram	PRCPRG
PrfCloseProfile	PRCPR
PrfCreateGroup	PRCGRP
PrfDestroyGroup	PRDGRP
PrfOpenProfile	PROPPR
PrfQueryDefinition	PRQDEF
PrfQueryProfile	PRQPR
PrfQueryProfileData	PRQPDA
PrfQueryProfileInt	PRQPIN
PrfQueryProfileSize	PRQPSZ
PrfQueryProfileString	PRQPST
PrfQueryProgramCategory	PRQPC
PrfQueryProgramHandle	PRQPHD
PrfQueryProgramTitles	PRQPTI
PrfRemoveProgram	PRRPRG
PrfReset	PRRES
PrfWriteProfileData	PRWPDA
PrfWriteProfileString	PRWPST

PrfAddProgram — Add Program

This call adds a program entry to the program list.

PrfAddProgram (hini, Details, Group, *Program***)**

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	The information in the user profile is updated
HINI_USERPROFILE	The information in the user profile is updated
Other	Initialization-file handle.

Details *(PROGDETAILS)* — input
Program details for the program to be added to the program list.

The initial window position and size information are contained within this parameter. If these are set to zero, the system generates a default starting size and position. For a non-OS/2 Version 1.2 application, these fields must always be set to zero. For OS/2 Version 1.2 applications, it is also recommended to set these fields to zero, because the values are device dependent.

The visibility attribute controls whether the program is visible in the program list (and therefore can be started by the user).

Group *(HPROGRAM)* — input
Handle of the group to which the program is to be added.

The group must not have the protected attribute.

NULL	Add to the default group (the first defined group)
Other	Group handle.

Program *(HPROGRAM)* — return
Program handle for the program added to the program list.

NULL	Error occurred
Other	Program handle.

Possible returns from WinGetLastError:

PMERR_INVALID_GROUP_HANDLE
PMERR_INVALID_TITLE
PMERR_NOT_IN_IDX
PMERR_NOT_CURRENT_PL_VERSION
PMERR_INSUFF_SPACE_TO_ADD
PMERR_INVALID_TARGET_HANDLE
PMERR_DUPLICATE_TITLE
PMERR_MEMORY_DEALLOCATION_ERR

Remarks

Program titles need not be unique, although duplicate titles within the same group are not allowed.

This call requires the existence of a message queue.

PrfChangeProgram — Change Program

This call replaces the information stored in the program list.

```
PrfChangeProgram (hini, Program, Details, Success)
```

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	The information in the user profile is updated
HINI_USERPROFILE	The information in the user profile is updated
Other	Initialization-file handle.

Program *(HPROGRAM)* — input
Handle of the program whose information is to be changed.

If this is a group handle, only the program type (visibility) and program title fields can be changed. If it is a program handle, then the group to which the program belongs must not have the protected attribute.

Details *(PROGDETAILS)* — input
Program details.

The replacement program details for the program.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The function can be used to change the details of a program in the program list, however the group associations for the program are not affected by this call.

None of the previous information stored in the program list, for this program, is retained.

PrfCloseProfile — Close Profile

This call indicates that a profile is no longer available for use.

PrfCloseProfile (hini, *Success*)

Parameters

hini *(HINI)* — input
Initialization-file handle.

After this call, the handle is no longer valid.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion.
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INI_FILE_IS_SYS_OR_USER
PMERR_INVALID_INI_FILE_HANDLE

Remarks

This call cannot be used to close the current user or system initialization files.

PrfCreateGroup — Create Group

This call creates a new program group entry in the program list.

PrfCreateGroup (hini, Title, Visibility, *Group*)

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	The information in the user profile is updated
HINI_USERPROFILE	The information in the user profile is updated
Other	Initialization-file handle.

Title *(STRL)* — input
Title of the new group.

This is displayed to the end-user and so should be readable and meaningful.

The title string must contain at least one nonblank character and must not contain "\".

Visibility *(BIT8)* — input
Visibility control.

Controls the visibility and other attributes of the new group:

SHE_VISIBLE	Group is visible (can be viewed by end-user).
SHE_INVISIBLE	Group is invisible.
SHE_UNPROTECTED	Group is not protected (default).
SHE_PROTECTED	Group is protected. This implies that a program within the group cannot be changed by use of the PrfChangeProgram call, neither can a program by added to or deleted from the group by the use of the PrfAddProgram call or the PrfRemoveProgram call, respectively. Also, the group cannot be destroyed by use of the PrfDestroyGroup call. A protected group is changed to an unprotected group by use of the PrfChangeProgram call.

Group *(HPROGRAM)* — return
Group handle for the newly-created group.

NULL	Error occurred
Other	Program-group handle.

Possible returns from WinGetLastError:

PMERR_INVALID_GROUP_HANDLE
PMERR_INVALID_TITLE
PMERR_NOT_IN_IDX
PMERR_NOT_CURRENT_PL_VERSION
PMERR_INSUFF_SPACE_TO_ADD
PMERR_MEMORY_DEALLOCATION_ERR

PrfCreateGroup – Create Group

Remarks

Because the newly created group is empty, the PrfAddProgram call must be used to add program entries to the group.

If the group already exists, the existing group handle is returned and no new group handle is created.

This call requires the existence of a message queue.

PrfDestroyGroup — Destroy Group

This call removes the definition of a specified group from the program list.

```
PrfDestroyGroup (hini, Group, Success)
```

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	The information in the user profile is updated
HINI_USERPROFILE	The information in the user profile is updated
Other	Initialization-file handle.

Group *(HPROGRAM)* — input
Group handle.

Handle of the group to be removed from the program list. The group must not have the protected attribute.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The associations between the group and its programs are destroyed.

It is not permitted to destroy the last group from the program list.

PrfOpenProfile – Open Profile

This call indicates that a file is available for use as a profile.

PrfOpenProfile (hab, FileName, *hini*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

FileName *(STRL)* – input
User-profile file name.

This must not be the same as the current user or system initialization file name.

hini *(HINI)* – return
Initialization-file handle.

This handle is used on other calls to manipulate the profile file.

NULL Error occurred
Other Initialization-file handle.

Possible returns from WinGetLastError:

PMERR_MEMORY_ALLOC
PMERR_OPENING_INI_FILE
PMERR_INI_FILE_IS_SYS_OR_USER

Remarks

A user profile and a system profile are opened by the system, either at start-up time, or (in the case of the user profile) as a result of a PrfReset call, and are always available. Their handles are HINI_USERPROFILE and HINI_SYSTEMPROFILE. Applications do not have to open or close the user profile or the system profile.

The handle returned is only valid for the process issuing the PrfOpenProfile call.

The PrfOpenProfile call can be used by an administrator's application that is creating or modifying a profile for a user.

It can also be used to create a back-up profile as follows:

- Use the enumerate form of PrfQueryProfileData to obtain a list of application names in the profile being backed up.
- Use the enumerate form of PrfQueryProfileData to obtain a list of key names for each of the application names.
- Use PrfQueryProfileData for each application name / key name pair to read the appropriate data.
- Use PrfWriteProfileData to write the data into the back-up profile.

PrfQueryDefinition — Query Definition

This call obtains the program details for a program or group.

PrfQueryDefinition (hini, Program, Details, MaxLen, *RetLen***)**

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

Program *(HPROGRAM)* — input
Handle of the program whose details are to be returned.

Details *(PROGDETAILS)* — output
Program details.

The strings referenced by this structure are placed after it in the buffer.

If the **Program** parameter is the handle of a group, only the **type** and **title** fields are significant.

MaxLen *(ULONG)* — input
Maximum length in bytes:

0	Return in the **RetLen** parameter the length required to contain the *PROGDETAILS* structure (and its concatenated strings).
Other	Length, in bytes, of data that can be returned in the **Details** parameter, including all of the concatenated strings.

RetLen *(ULONG)* — return
Length of returned data:

0	Error occurred
Other	Length of data actually returned in the **Details** parameter (or length that is needed if **MaxLen** is 0).

Possible returns from WinGetLastError:

PMERR_NOT_IN_IDX
PMERR_INVALID_PROGRAM_HANDLE
PMERR_NOT_CURRENT_PL_VERSION
PMERR_MEMORY_ALLOCATION_ERR
PMERR_MEMORY_DEALLOCATION_ERR
PMERR_BUFFER_TOO_SMALL

PrfQueryDefinition – Query Definition

Remarks

This call is normally used by passing a **MaxLen** parameter with a value of 0 in order to get the total amount of space required, then allocating a buffer of the required length. The buffer can then be passed to a second call of this function.

Programming Note: If the length of the **Details** parameter is only sufficient to contain a *PROGDETAILS* structure, this call fails, because there is insufficient space to copy the concatenated strings.

If the **Program** parameter refers to a program rather than a group, the data returned in the **Details** parameter is in a format that can be used by the PrfAddProgram or PrfChangeProgram calls.

This call requires the existence of a message queue.

PrfQueryProfile — Query Profile

This call returns a description of the current user and system profiles.

PrfQueryProfile (hab, Profile, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Profile *(PRFPROFILE)* — output
Profile names structure.

The **userlen** and the **syslen** parameters of the *PRFPROFILE* data structure are set to the lengths of the respective file names, even if truncation occurs. If these fields are initialized to zero by the application, then the **username** and **sysname** parameters are not inspected, and the application can then determine the sizes of the buffers required to hold the names on a second call. Otherwise, the **username** and **sysname** parameters must point to reserved areas of memory, and the **userlen** and **syslen** parameters must indicate the sizes of those areas.

If the **username** or the **sysname** parameter is NULL, then there is no defined user or system profile, respectively.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion.
FALSE Error occurred, or there was insufficient space to record the names, which have been truncated.

PrfQueryProfileData — Query Profile Data

This call returns a string of binary data from the specified profile.

PrfQueryProfileData (hini, App, Key, Value, Size, *Success*)

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

App *(STRL)* — input
Application name.

The name of the application for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

If this parameter is NULL, this call enumerates all the application names present in the profile and returns the names as a list in the **Value** parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, the **Size** parameter contains the total length of the list excluding the final NULL character.

Key *(STRL)* — input
Key name.

The name of the key for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

If this parameter is NULL, and if **App** is not equal to NULL, this call enumerates all key names associated with the named application and returns the key names, but not their values, as a list in the **Value** parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, the **Size** parameter contains the total length of the list **excluding** the final NULL character.

Value *(STORAGE)* — output
Value data.

A buffer in which the value corresponding to the key name is returned. The returned data is not null terminated, unless the value data is explicitly null terminated within the file. This call handles binary data.

Size *(LENGTH4)* — input/output
Size of value data.

This is the size of the buffer specified by the **Value** parameter. If the call is successful, this is overwritten with the number of bytes copied into the buffer.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_NOT_IN_IDX
PMERR_CAN_NOT_CALL_SPOOLER

Remarks

This call returns a string of binary data from the profile. The call searches the file for a key matching the name specified by the **Key** parameter, under the application heading specified by the **App** parameter.

Enumeration can be performed in exactly the same way as in the PrfQueryProfileString call. The enumeration returns application or key names irrespective of whether the data concerned is written with the PrfWriteProfileString call or the PrfWriteProfileData call.

This call returns data that is written to the file using either the PrfWriteProfileString call or the PrfWriteProfileData call.

If the **App** parameter is NULL, this call enumerates all application names and constructs in the **Value** parameter a list of application names. Each application name in the list is terminated with a null character. The last string in the list is terminated with two null characters. This call returns the length of the list, up to, but not including, the final null. If the enumerated application names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with two bytes of zeros, and the **Success** parameter is set to FALSE. In this case, **Key** is ignored.

If the **App** parameter is valid and if the **Key** is NULL, this call enumerates all key names associated with the **App** parameter by constructing in the **Value** parameter a list of key names. Each key name in the list is terminated with a null character. The last string in the list is terminated with two null characters. This call returns the length of the list, up to, but not including, the final null. If the enumerated key names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with two bytes of zeros, and the **Success** parameter is set to FALSE.

This call requires the existence of a message queue.

PrfQueryProfileInt – Query Profile Integer

This call returns an integer value from the specified profile.

PrfQueryProfileInt (hini, App, Key, Default, *Result*)

Parameters

hini *(HINI)* – input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

App *(STRL)* – input
Application name.

The name of the application for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

Key *(STRL)* – input
Key name.

The name of the key for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

Default *(SHORT)* – input
Default value.

This value is returned in **Result**, if the key defined by **Key** cannot be found in the initialization file.

Result *(SHORT)* – return
Key value specified in the initialization file.

The value of the key specified by **Key** in the initialization file.

If the value corresponding to the key is not an integer, **Result** is 0.

If the key-name value is a series of digits followed by non-numeric characters, **Result** contains the value of the digits only. For example, "KeyName=102abc" causes the value 102 to appear in **Result**.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_NOT_IN_IDX
PMERR_CAN_NOT_CALL_SPOOLER

Remarks

This call returns an integer value from the profile. The call searches the file for a key matching the name specified by the **Key** parameter, under the application heading specified by the **App** parameter. When an integer is stored as a text string using the PrfWriteProfileString call, for example, "123," the returned value is the number, 123. The call returns **Default** if the application-name/key-name pair cannot be found.

Note: The search is case-dependent.

This call requires the existence of a message queue.

PrfQueryProfileSize — Query Profile Size

This call obtains the size, in bytes, of the value of a specified key for a specified application in the profile.

PrfQueryProfileSize (hini, App, Key, DataLen, *Success***)**

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

App *(STRL)* — input
Application name.

The name of the application for which the profile data is required.

If the **App** parameter is NULL, then the **DataLen** parameter returns the length of the buffer required to hold the enumerated list of application names, as returned by the PrfQueryProfileString call when its **App** parameter is NULL. In this case, the **Key** parameter is ignored.

Key *(STRL)* — input
Key name.

The name of the key for which the size of the data is to be returned.

If the **Key** parameter is NULL, and if the **App** parameter is not NULL, the **DataLen** returns the length of the buffer required to hold the enumerated list of key names for that application name, as returned by the PrfQueryProfileString call when its **Key** parameter is NULL, and its **App** parameter is not NULL.

DataLen *(LENGTH4)* — output
Data length.

This parameter is the length of the value data related to the **Key** parameter. If an error occurs, this parameter is undefined.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_NOT_IN_IDX
PMERR_CAN_NOT_CALL_SPOOLER

PrfQueryProfileSize — Query Profile Size

Remarks

The **App** parameter and **Key** parameter are case sensitive and must match the names stored in the file exactly. There is no case-independent searching.

This call can be used before using the PrfQueryProfileString call, to allocate space for the returned data.

No distinction is made between data that is written using the PrfWriteProfileData call and the PrfWriteProfileString call.

This call requires the existence of a message queue.

PrfQueryProfileString — Query Profile String

This call retrieves a string from the specified profile.

PrfQueryProfileString (hini, App, Key, Default, ProfileString, MaxLen, *RetLen***)**

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

App *(STRL)* — input
Application name.

The name of the application for which the profile data is required.

The search on application name is always case-dependent. Names starting with the characters "PM_" are reserved for system use.

If this parameter is NULL, this call enumerates all the application names present in the profile and returns the names as a list in the **ProfileString** parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the **RetLen** parameter contains the total length of the list **excluding** the final NULL character.

Key *(STRL)* — input
Key name.

The name of the key for which the profile data is returned.

The search on key name is always case-dependent.

If this parameter equals NULL, and if the **App** parameter is not equal to NULL, this call enumerates all key names associated with the named application and returns the key names (not their values) as a list in the **ProfileString** parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the **RetLen** parameter contains the total length of the list **excluding** the final NULL character.

Default *(STRL)* — input
Default string.

The string that is returned in the **ProfileString** parameter, if the key defined by the **Key** parameter cannot be found in the profile.

If the pointer to this parameter is passed as NULL, then nothing is copied into the **Key** parameter if key cannot be found. In this case, **RetLen** is returned as zero.

ProfileString *(STORAGE)* — output
Profile string.

The text string obtained from the profile for the key defined by the **Key** parameter.

MaxLen *(LENGTH4)* — input
Maximum string length.

The maximum number of characters that can be put into the **ProfileString** parameter, in bytes. If the data from the profile is longer than this, it is truncated.

PrfQueryProfileString — Query Profile String

RetLen *(LENGTH4)* — return
String length returned.

The actual number of characters (including the null termination character) returned in the **ProfileString** parameter, in bytes.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_BUFFER_TOO_SMALL
PMERR_NOT_IN_IDX
PMERR_CAN_NOT_CALL_SPOOLER
PMERR_INVALID_ASCIIZ

Remarks

The call searches the profile for a key matching the name specified by the **Key** parameter under the application heading specified by the **App** parameter. If the key is found, the corresponding string is copied. If the key does not exist, the default character string, specified by the **Default** parameter, is copied.

If the enumerated application names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with two bytes of zeros, and the **RetLen** parameter is set to the number of bytes copied into the **ProfileString** parameter. In this instance, the **Key** parameter is ignored.

Programming Note: If the enumeration cannot be performed for any reason, the default character string is **not** copied.

This call returns the length of the list, up to, but not including, the final null. If the enumerated key names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with two bytes of zeros, and the **RetLen** parameter is set to the number of bytes copied into the **ProfileString** parameter.

This call is case-dependent; thus the strings in the **App** parameter and the **Key** parameter must match exactly. This avoids any code-page dependency and it is up to the application storing the data to do any case-independent matching.

The enumeration call does not distinguish between data written with the PrfWriteProfileString call and the PrfWriteProfileData call.

This call requires the existence of a message queue.

PrfQueryProgramCategory — Query Program Category

This call returns the program category of the named program.

PrfQueryProgramCategory (hini, Exe, *Category*)

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

Exe *(STRL)* — input
Executable-file name.

Category *(PROGCATEGORY)* — return
Program category.

This value corresponds to one of the values of the *PROGCATEGORY* data type, namely, PROG_DEFAULT, PROG_REAL, PROG_WINDOWABLEVIO, PROG_FULLSCREEN, or PROG_PM. See *PROGTYPE* for more information.

PrfQueryProgramHandle — Query Program Handle

This call obtains the program handles that match an executable file name.

```
PrfQueryProgramHandle (hini, Exe, ProgArray, MaxLen, Count, RetLen)
```

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

Exe *(STRL)* — input
Executable-file name.

ProgArray *(HPROGARRAY)* — output
Program handles.

One for each match found.

MaxLen *(COUNT4)* — input
Maximum number of program handles.

This is the number of bytes of data that can be placed into the **ProgArray** parameter.

Count *(COUNT4)* — output
Number of program handles returned.

This is the number of program handles actually returned in the **ProgArray** parameter.

RetLen *(LENGTH4)* — return
Number of bytes returned.

0	Error occurred
Other	Number of bytes required or returned in the **ProgArray** parameter.

Remarks

This call allows the caller to determine if, and where, a given executable file is referenced in the program list.

PrfQueryProgramTitles — Query Program Titles

This call obtains information about a program or a group defined in the program list.

PrfQueryProgramTitles (hini, Group, Titles, MaxLen, Count, *Retlen*)

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

Group *(HPROGRAM)* — input
Handle of the program or group whose information is to be returned.

NULL	Return the information for the default group only. The default group is the one to which a program is added by the PrfAddProgram call with its **Group** parameter set to NULL.
SGH_GROUP	Return the information for all groups.
Other	Handle of the program or group whose information is to be returned.

Titles *(PROGTITLE*Count)* — output
Program information buffer.

An area of storage into which the program information is returned. This is an array of *PROGTITLE* data structures, followed by the title strings.

If the **Group** parameter is a program handle rather than a group handle, a single record of information is returned, that is the data for that program alone.

MaxLen *(LENGTH4)* — input
Buffer length.

The total length of **Titles**, in bytes.

0	Return in the **Retlen** parameter the length required to contain the *PROGTITLE* structures (and their associated strings).
Other	Buffer length; a length less than the size of a *PROGTITLE* data structure is invalid.

Count *(COUNT4)* — output
Number of structures returned.

The total number of *PROGTITLE* data structures that are in the group.

Retlen *(LENGTH4)* — return
Length returned.

NULL	Error occurred
Other	Total length of *PROGTITLE* data that is passed back, or that is required if **MaxLen** is 0.

Possible returns from WinGetLastError:

PMERR_NOT_IN_IDX
PMERR_INVALID_GROUP_HANDLE
PMERR_NOT_CURRENT_PL_VERSION
PMERR_INVALID_TARGET_HANDLE
PMERR_BUFFER_TOO_SMALL

PrfQueryProgramTitles – Query Program Titles

Remarks

Information about all programs in a group is returned in a single operation as an array of entries, one for each program in the group.

This call requires the existence of a message queue.

PrfRemoveProgram — Remove Program Definition

This call removes the definition of a program from the program list.

```
PrfRemoveProgram (hini, Program, Success)
```

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	The information in the user profile is updated
HINI_USERPROFILE	The information in the user profile is updated
Other	Initialization-file handle.

Program *(HPROGRAM)* — input
Program handle.

Program handle for the program to be removed from the program list. No group containing the program must have the protected attribute.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

References to the program from all groups are removed.

This function cannot be used to remove a program group definition. This is achieved by the use of the PrfDestroyGroup call.

PrfReset – Reset Presentation Manager

This call defines which files are to be used as the user and system profiles.

```
PrfReset (hab, Profile, Success)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

Profile *(PRFPROFILE)* – input
Profile-names structure.

This contains the name(s) of the file(s) to be used as the new Presentation Manager profile files.

The name of the system profile cannot be changed. It must be the name of the current system profile as returned by PrfQueryProfile.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_OPENING_INI_FILE

Remarks

This call caused the workstation to use different profiles. When the workstation is initialized, the names of the user and system profiles are taken from the PROTSHELL statement specified in CONFIG.SYS. The PrfReset allows the profiles to be changed during operation of the workstation, for example by a logon application controlling multiple consecutive users of the system.

After the PrfReset call completes, the system has a new set of preferences (for example screen colors), a new start-up list, and new spooler parameters.

The PrfReset call broadcasts the PL_ALTERED message, which must be processed by all applications that read their default settings from the user or system profiles.

For example, consider the control panel and logon applications. On receipt of a PL_ALTERED message, these carry out the following:

- The color settings are read from the new profiles, and screen colors (and palettes) refreshed.
- The country information, for example the date and time format, is read from the new profiles.
- Other preferences, for example those that affect the operations of the alarm and the mouse, are also updated with the settings held in the new profiles.

PrfWriteProfileData — Write Profile Data

This call writes a string of binary data into the specified profile.

PrfWriteProfileData (hini, App, Key, Value, Size, *Success*)

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	The information is written to the user profile
HINI_USERPROFILE	The information is written to the user profile
HINI_SYSTEMPROFILE	The information is written to the system profile
Other	Initialization-file handle.

App *(STRL)* — input
Application name.

The case-dependent name of the application for which profile data is to be written. Names starting with the characters "PM_" are reserved for system use.

Key *(STRL)* — input
Key name.

The case-dependent name of the key for which profile data is to be written.

This parameter can be NULL in which case **all** the **Key / Value** pairs associated with **App** are deleted.

Value *(STORAGE)* — input
Value data.

This is the value of the **Key / Value** pair that is written to the profile. It is **not** zero-terminated, and its length is given by the **Size** parameter.

If this parameter is NULL, the string associated with the **Key** parameter is deleted.

Size *(LENGTH4)* — input
Size of value data.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_CAN_NOT_CALL_SPOOLER

Remarks

Because of the binary nature of the data, the input data is not zero-terminated. The length provided is the only way to identify the length of the data.

This call requires the existence of a message queue.

PrfWriteProfileString — Write Profile String

This call writes a string of character data into the specified profile.

```
PrfWriteProfileString (hini, App, Key, Value, Success)
```

Parameters

hini *(HINI)* — input
Initialization-file handle.

HINI_PROFILE	The information is written to the user profile
HINI_USERPROFILE	The information is written to the user profile
HINI_SYSTEMPROFILE	The information is written to the system profile
Other	Initialization-file handle.

App *(STRL)* — input
Application name.

The case-dependent name of the application for which profile data is to be written. Names starting with the characters "PM_" are reserved for system use.

Key *(STRL)* — input
Key name.

The case-dependent name of the key for which profile data is to be written.

This parameter can be NULL, in which case **all** the **Key / Value** pairs associated with the **App** parameter are deleted.

Value *(STRL)* — input
Text string.

This is the value of the **Key / Value** pair that is written to the profile.

If this parameter is NULL, the string associated with the **Key** is deleted (that is, the entry is deleted).

If this parameter is not NULL, the string is used as the value of the **Key / Value** pair, even if the string has zero length.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_CAN_NOT_CALL_SPOOLER

Remarks

If there is no application field in the file that matches the **App**, a new application field is created before the **Key / Value** entry is made.

If the keyname does not exist for the application, a new **Key / Value** entry is created for that application. If the **Key** already exists in the file, the existing value is overwritten.

This call requires the existence of a message queue.

Chapter 7. Spooler Function Calls

The following table shows the Spooler (Spl) function calls in alphabetic order according to their C/MASM name; the COBOL/FORTRAN name is also given.

C/MASM name	COBOL/FORTRAN name
SplQmAbort	SMABT
SplQmClose	SMCLS
SplQmEndDoc	SMEDOC
SplQmOpen	SMOPEN
SplQmStartDoc	SMSDOC
SplQmWrite	SMWRT
SplQpInstall	SPINST
SplQpQueryDt	SPQDT

SplQmAbort –
Spooler Queue Manager Abort

This call stops the generation of the spool file(s). It automatically closes the Print Manager (see SplQmClose).

This function can be called from either ring 2 or ring 3.

```
SplQmAbort (hspl, Success)
```

Parameters

hspl *(HSPL)* – input
Spooler handle.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

SplQmClose – Spooler Queue Manager Close

This call corresponds to the DevCloseDC call: it closes the Print Manager.

This function can be called from either ring 2 or ring 3.

SplQmClose (hspl, *Success***)**

Parameters

hspl *(HSPL)* – input
Spooler handle.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

SplQmEndDoc –
Spooler Queue Manager End Document

This call corresponds to the DevEscape (DEVESC_ENDDOC) call: it ends a print job, and returns **Job**, a unique number to identify the job.

SplQmEndDoc (hspl, *Job***)**

Parameters

hspl *(HSPL)* – input
Spooler handle.

Job *(USHORT)* – return
Job identifier:

Nonzero	Jobid (1 through 65 535)
SPL_ERROR	Error.

Remarks

The print-job identifier is displayed to the user by the spooler while this job is on the queue, and while it is being printed.

This function can be called from either ring 2 or ring 3.

SplQmOpen – Spooler Queue Manager Open

This call corresponds to the DevOpenDC call: it opens the Print Manager for generating a print job.

SplQmOpen (Token, Count, Data, *hspl*)

Parameters

Token *(STRL)* – input
A token (nickname) that identifies spooler information.

This information is held in the initialization file, and is the same as that in **Data**; any that is obtained from **Data**; overrides the information obtained using **Token**.

If **Token** is specified as "*", then no device information is taken from the initialization file.

Presentation Manager behaves as if "*" is specified, but it allows any string to be specified.

Count *(LONG)* – input
Number of items.

This is the number of items present in the **Data** supplied. This can be shorter than the full list, if omitted items are irrelevant, or supplied from **Token** or elsewhere.

Data *(QMOPENDATA)* – input
Open parameters.

hspl *(HSPL)* – return
Spooler handle:

Nonzero	Spooler handle
SPL_ERROR	Error.

Remarks

This function can be called from either ring 2 or ring 3. The pointer parameter passed can be either a ring 2 or ring 3 pointer.

SplQmStartDoc –
Spooler Queue Manager Start Document

This call corresponds to the DevEscape (DEVESC_STARTDOC) call; it starts a print job.

```
SplQmStartDoc (hspl, DocName, Success)
```

Parameters

hspl *(HSPL)* – input
Spooler handle.

DocName *(STRL)* – input
Document name.

This is part of the job description which is displayed to the end user by the spooler.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This call signifies the start of a print job. It allows the application to specify a document name to be associated with the print job.

Multiple print jobs can be generated, within a single queue manager open, by bracketing each job with SplQmStartDoc and SplQmEndDoc.

This function can be called from either ring 2 or ring 3. The pointer parameter passed can be either a ring 2 or ring 3 pointer.

SplQmWrite – Spooler Queue Manager Write

This call writes a buffer of data to the spool file for the print job.

SplQmWrite (hspl, Count, Data, *Success***)**

Parameters

hspl *(HSPL)* – input
Spooler handle.

Count *(LONG)* – input
Length in bytes.

This is the length of **Data**; it must not be greater than 65 535. Data that is longer than this must be written by two or more calls.

Data *(BUFFER)* – input
Buffer of data to be written to the spool file.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This function can be called from either ring 2 or ring 3. The pointer to the **Data** parameter passed can be either a ring 2 or ring 3 pointer.

SplQpInstall – Spooler Queue Processor Install

This call performs any installation work that is required. It is called by the control panel when the queue processor is installed.

SplQpInstall (hwnd, *Success*)

Parameters

hwnd *(HWND)* – input
Window handle.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The processor holds a conversation with the user to obtain any processor-dependent options. These are then stored in the initialization file.

SplQpQueryDt – Spooler Queue Processor Query Data Type

This call returns a list of the data types supported by the queue processor.

SplQpQueryDt (Count, Datatypes, *Success***)**

Parameters

Count *(LONG)* – input/output
Maximum number of data types that can be returned.

If this is set to zero, the number of data types supported is returned in **Count**, and **Datatypes** is not updated. If nonzero, this is updated to the number actually returned in **Datatypes**.

Datatypes *(STRL****Count)*** – output
An array containing the data types supported.

Each element in the array is a data type name.

The data types are:

PM_Q_STD Standard format
PM_Q_RAW Raw format
Other Application-defined data types.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The application can call the function first with **Count** set to zero, to find out how much storage is needed for the **Datatypes** array. Having allocated the storage, the application then calls the function a second time for the data to be filled in.

Chapter 8. Advanced Video Function Calls

The following table shows the Advanced Video (AVIO) function calls in alphabetic order according to their C/MASM name.

For Details of the Base VIO calls see the Operating System/2 *Control Program Programming Reference*.

C/MASM name	COBOL/FORTRAN name
VioAssociate	not used
VioCreateLogFont	not used
VioCreatePS	not used
VioDeleteSetId	not used
VioDestroyPS	not used
VioGetDeviceCellSize	not used
VioGetOrg	not used
VioQueryFonts	not used
VioQuerySetIds	not used
VioSetDeviceCellSize	not used
VioSetOrg	not used
VioShowPS	not used

VioAssociate – Vio Associate

This call associates a VIO presentation space with a device context.

VioAssociate (hdc, hvps, *Error*)

Parameters

hdc *(HDC)* – input
Device-context handle.

If NULL, a disassociation occurs.

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0	NO_ERROR
436	ERROR_VIO_INVALID_HANDLE
495	ERROR_VIO_NOT_PRES_MGR_SG
498	ERROR_VIO_SEE_ERROR_LOG
499	ERROR_VIO_ASSOCIATED_DC

Remarks

Subsequent VioShowPS or VioShowBuf (see Operating System/2 *Control Program Programming Reference*) calls direct output to this device context.

If a null handle is supplied for the device context, the presentation space is disassociated from its currently associated device context. An associated presentation space or device context cannot be associated with another device context or presentation space, respectively.

A screen device context is the only kind of device context that can be associated with a VIO presentation space.

VioCreateLogFont – Vio Create Logical Font

This call specifies a logical definition of a font that the application wants to use in a VIO presentation space.

VioCreateLogFont (attrs, lcid, Name, hvps, *Error***)**

Parameters

attrs *(FATTRS)* – input
Attributes of font.

The required attributes of the font.

lcid *(LONG)* – input
Local identifier.

The local identifier that the application uses to refer to this font. It must be in the range **1** through **3**.

It is an error if **lcid** is already in use to refer to a font.

Name *(STR8)* – input
Logical-font name.

An 8-character name that may be used to describe the logical font.

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
436	ERROR_VIO_INVALID_HANDLE
495	ERROR_VIO_NOT_PRES_MGR_SG
498	ERROR_VIO_SEE_ERROR_LOG

Remarks

The system uses 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 VioQueryFonts.

The local identifier (**lcid**) that the application needs to use to reference this logical font in the VIO presentation space is also specified.

The physical fonts that can be used to satisfy this call can either be provided by the device driver, made publicly available by loading from the control panel, or loaded (from the same process) with GpiLoadFonts.

Only fixed-pitch fonts of the correct cell size, as defaulted or set by VioSetDeviceCellSize can be used. They may be publicly loaded fonts, fonts loaded by GpiLoadFonts, or ones supplied by the device driver.

VioCreatePS –
Vio Create Presentation Space

This call creates an AVIO presentation space.

VioCreatePS (hvps, depth, width, Format, Attrs, Reserved, *Error*)

Parameters

hvps *(HVPS)* – output
VIO presentation-space handle.

This is the handle of the newly-created AVIO presentation space. It must be passed on all subsequent VIO calls for this presentation space.

Note: This is a 16-bit handle, unlike other Presentation Manager handles.

depth *(SHORT)* – input
Presentation-space depth.

Depth of the presentation space required in character cell units. The maximum allowed is 255.

width *(SHORT)* – input
Presentation-space width.

Width of the presentation space required in character cell units. The maximum allowed is 255.

Format *(SHORT)* – input
Presentation-space format.

This identifies the format of the attribute byte(s) in the presentation space. **0** is the only defined format; for this, the presentation-space layout is:

Attrs = 1 (CGA)

Code point	
Base attributes	(bits 7–4) background color (bits 3–0) foreground color (16 colors for each)

Attrs = 3 (Extended)

Code point	
Base attributes	(bits 7–4) background color (bits 3–0) foreground color (16 colors for each)
Extended attrs	(bit 7) underscore (bit 6) reverse video (bits 1–0) font local identifier (0, 1, 2, or 3) Other bits reserved, should be zero
Reserved byte	This byte must not be used by an application.

The encoding for foreground and background colors is the same as for the Enhanced Graphics Adapter.

Attrs *(SHORT)* – input
Attribute bytes per character.

The number of attribute bytes in the presentation space per character cell. This must be **1** or **3**.

Reserved *(HVPS)* – input
Reserved (must be zero).

Error *(USHORT)* – return
Error indicator:

0	NO_ERROR
421	ERROR_VIO_INVALID_PARMS
495	ERROR_VIO_NOT_PRES_MGR_SG
497	ERROR_VIO_NO_MORE_HANDLES
498	ERROR_VIO_SEE_ERROR_LOG

Remarks

The presentation-space size (**width** * **depth** * (**Attrs** + 1)) must not exceed 64KB.

There is a limit of 32 on the number of VIO presentation spaces per process.

VioDeleteSetId – Vio Delete Set Id

This call deletes the logical font (or all logical fonts) and makes any deleted font unavailable for use.

```
VioDeleteSetId (lcid, hvps, Error)
```

Parameters

lcid *(LONG)* – input
Local identifier.

The local identifier (lcid) for the object, as specified in VioCreateLogFont. If LCID_ALL is specified, **all** logical fonts are deleted.

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0 NO_ERROR
421 ERROR_VIO_INVALID_PARMS
436 ERROR_VIO_INVALID_HANDLE
495 ERROR_VIO_NOT_PRES_MGR_SG
498 ERROR_VIO_SEE_ERROR_LOG

Remarks

After this call has completed, the **lcid** is freed and is available for reuse.

VioDestroyPS – Vio Destroy Presentation Space

This call destroys the VIO presentation space.

```
VioDestroyPS (hvps, Error)
```

Parameters

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0	NO_ERROR
436	ERROR_VIO_INVALID_HANDLE
495	ERROR_VIO_NOT_PRES_MGR_SG
498	ERROR_VIO_SEE_ERROR_LOG
499	ERROR_VIO_ASSOCIATED_DC

Remarks

The presentation space must be dissociated (see VioAssociate) before this call is issued.

The VIO presentation-space handle is invalid after this call.

VioGetDeviceCellSize – Vio Get Device Cell Size

This call gets the current device cell size.

VioGetDeviceCellSize (Height, Width, hvps, *Error*)

Parameters

Height *(SHORT)* – output
Current device cell height in pels.

Width *(SHORT)* – output
Current device cell width in pels.

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0 NO_ERROR
436 ERROR_VIO_INVALID_HANDLE
495 ERROR_VIO_NOT_PRES_MGR_SG

VioGetOrg – Vio Get Origin

This call gets the origin; that is, how the VIO presentation space buffer maps to the window. See VioSetOrg.

VioGetOrg (Row, Column, hvps, *Error***)**

Parameters

Row *(SHORT)* – output
Row number.

This is the row number of the cell currently mapped to the top left hand corner of the window (first is zero).

Column *(SHORT)* – output
Column number.

This is the column number of the cell currently mapped to the top left hand corner of the window (first is zero).

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0 NO_ERROR
436 ERROR_VIO_INVALID_HANDLE
495 ERROR_VIO_NOT_PRES_MGR_SG

VioQueryFonts – Vio Query Fonts

This call returns a record providing details of the fonts that match the specified facename.

VioQueryFonts (Remfonts, Metrics, MetricsLength, Fonts, Facename, Options, hvps, *Error***)**

Parameters

Remfonts *(LONG)* – output
Number of fonts for which information is not returned.

Metrics *(FONTMETRICS****Fonts)*** – output
Font metrics.

In this buffer are returned the font metrics of up to **Fonts** matching fonts. For each font, no more data than **MetricsLength** is returned.

MetricsLength *(LONG)* – input
The length of each metrics record to be returned.

The data area **Metrics** must be **Fonts** multiplied by **MetricsLength** in length.

Fonts *(LONG)* – input/output
Number of fonts.

Number of fonts for which the application requires the metrics.

The number of fonts that were actually returned is returned in this variable.

Facename *(STRL)* – input
Facename of the fonts of interest.

Options *(BIT32)* – input
Enumeration options.

This controls which fonts are to be enumerated:

QF_PUBLIC Enumerate public fonts
QF_PRIVATE Enumerate private fonts.

If both options are required, the values should be ORed together.

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0 NO_ERROR
436 ERROR_VIO_INVALID_HANDLE
495 ERROR_VIO_NOT_PRES_MGR_SG
498 ERROR_VIO_SEE_ERROR_LOG

Remarks

By inspecting the returned data, the application may choose which of the available fonts is most appropriate for its requirements. If necessary, it can force selection of a particular font by specifying a particular value, being one of the **Match** values returned by VioQueryFonts. This is the same as the **Match** value in the **attrs** structure for VioCreateLogFont.

By specifying **Fonts** as **0**, and then looking at the value returned in **Remfonts**, an application can determine how many fonts there are which match the **Facename**.

All sizes are returned in pel coordinates.

VioQuerySetIds – Vio Query Set Identifiers

This call queries all logical fonts loaded with VioCreateLogFont.

VioQuerySetIds (lcids, Names, Types, count, hvps, *Error***)**

Parameters

lcids *(LONG*count)* – output
Local identifiers.

An array in which the local identifier (lcid) values are returned.

Names *(STR8)* – output
Font names.

An array of **count** elements, into which the 8-character names associated with the logical fonts is returned.

Types *(LONG*count)* – output
Object types.

Elements indicate that the corresponding **lcids** element refers to a logical font.

LCIDT_FONT Font object.

count *(LONG)* – input
The number of objects to be queried.

The maximum number of objects in use for a VIO presentation space is 3.

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0 NO_ERROR
436 ERROR_VIO_INVALID_HANDLE
495 ERROR_VIO_NOT_PRES_MGR_SG
498 ERROR_VIO_SEE_ERROR_LOG

Remarks

Information about the first **count** objects is returned. If there are fewer than **count**, elements of **Types** and **lcids** for the remainder are set to zero.

This is the VIO equivalent of GpiQuerySetIds.

VioSetDeviceCellSize – Vio Set Device Cell Size

This call sets the device cell size.

VioSetDeviceCellSize (Height, Width, hvps, *Error*)

Parameters

Height *(SHORT)* – input
The required device cell height in pels.

Width *(SHORT)* – input
The required device cell width in pels.

hvps *(HVPS)* – input
VIO presentation space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0	NO_ERROR
436	ERROR_VIO_INVALID_HANDLE
495	ERROR_VIO_NOT_PRES_MGR_SG
498	ERROR_VIO_SEE_ERROR_LOG

Remarks

The cell sizes supported are as follows:

Note: The parameters are in the order **Width * Height.**

Where more than one size is supported, the first one is the default.

CGA	8*8
EGA	8*12, 8*8
VGA	8*14, 8*8
8514/A	8*17, 7*15, 7*25, 8*8, 12*16, 12*20, 12*22, 12*30

It is not an error to attempt to set a cell size other than one of those supported. If this happens on a device which supports more than one cell size, the larger size is selected, if both dimensions requested are greater than, or equal to, the actual cell size. Otherwise, the smaller size is selected. VioGetDeviceCellSize can be issued to determine which size has actually been selected.

Any logical fonts that the application has created (with VioCreateLogFont) must be recreated after changing the cell size.

VioSetOrg – Vio Set Origin

This call sets the origin; that is, how the VIO presentation space buffer maps to the window.

VioSetOrg (Row, Column, hvps, *Error***)**

Parameters

Row *(SHORT)* – input
Row number.

The row number of the cell to be mapped to the top left hand corner of the window (first is zero).

Column *(SHORT)* – input
Column number.

The column number of the cell to be mapped to the top left hand corner of the window (first is zero).

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0 NO_ERROR
358 ERROR_VIO_ROW
359 ERROR_VIO_COL
436 ERROR_VIO_INVALID_HANDLE
495 ERROR_VIO_NOT_PRES_MGR_SG
498 ERROR_VIO_SEE_ERROR_LOG

Remarks

This call can be used when the presentation space is larger than the window size, to control which area of the presentation space is visible. It does not itself cause any output to occur, and is normally followed by a VioShowPS or VioShowBuf call.

VioShowPS – Vio Show Presentation Space

This call updates the display with the VIO presentation space for a rectangle.

VioShowPS (Depth, Width, offCell, hvps, *Error***)**

Parameters

Depth *(SHORT)* – input
The depth of the rectangle in character cell units.

Width *(SHORT)* – input
The width of the rectangle in character cell units.

offCell *(SHORT)* – input
Cell offset.

Identifies the cell of the top left hand corner of the rectangle that has been updated since the last VioShowPS. The top left cell is offset 0.

hvps *(HVPS)* – input
VIO presentation-space handle.

This is returned by VioCreatePS.

Error *(USHORT)* – return
Error indicator:

0 NO_ERROR
421 ERROR_VIO_INVALID_PARMS
436 ERROR_VIO_INVALID_HANDLE
495 ERROR_VIO_NOT_PRES_MGR_SG

Chapter 9. Window Function Calls

Window Calls by Functional Area

The following table shows how all the Window (WIN) calls are related within functional areas. The calls are in alphabetic order within these areas according to their C/MASM name; the COBOL/FORTRAN name is also given.

C/MASM name	COBOL/FORTRAN name
Accelerators	
WinCopyAccelTable	WMCAT
WinCreateAccelTable	WMCACT
WinDestroyAccelTable	WMDACT
WinLoadAccelTable	WMLACT
WinQueryAccelTable	WMQACC
WinSetAccelTable	WMSACC
WinTranslateAccel	WMTAC
Alarms	
WinAlarm	WMALRM
WinFlashWindow	WIFLSH
WinMessageBox	WCMBOX
Atom Manager	
WinAddAtom	WUADAT
WinCreateAtomTable	WMCATT
WinDeleteAtom	WMDELA
WinDestroyAtomTable	WMDATT
WinFindAtom	WMFNAT
WinQueryAtomLength	WMQALN
WinQueryAtomName	WMQANM
WinQueryAtomUsage	WMQAUS
WinQuerySystemAtomTable	WMQSAT
Clipboard	
WinCloseClipbrd	WICCBD
WinEmptyClipbrd	WIECB
WinEnumClipbrdFmts	WIECF
WinOpenClipbrd	WIOPCB
WinQueryClipbrdData	WIQCBD
WinQueryClipbrdFmtInfo	WIQCBF
WinQueryClipbrdOwner	WIQCBO
WinQueryClipbrdViewer	WIQCBV
WinSetClipbrdData	WISCBD
WinSetClipbrdOwner	WISCBO

C/MASM name	COBOL/FORTRAN name
WinSetClipbrdViewer	WISCBV
COBOL/FORTRAN Support	
WinCreateDataStructure	WUCDS
WinDestroyDataStructure	WUDDS
WinFreeMsg	WUFMSG
WinGetDlgMsg	WCGETD
WinModifyDataStructure	WUMDS
WinQueryBits	WUQBIT
WinQueryBitsUnderMask	WUQBM
WinQueryDataStructure	WUQDS
WinQueryValue	WUQVAL
WinSetBits	WUSBIT
WinSetBitsUnderMask	WUSBM
WinSetValue	WUSVAL
Coordinate Mapping	
WinMakePoints	not used
WinMapDlgPoints	WCMDP
WinMapWindowPoints	WIMAPW
Cursor	
WinCreateCursor	WMCCUR
WinDestroyCursor	WMDCUR
WinQueryCursorInfo	WMQCUR
WinShowCursor	WISCUR
Dialog Boxes	
WinCreateDlg	WCCRD
WinDefDlgProc	WCDDP
WinDismissDlg	WCDELD
WinDlgBox	WCLMD
WinLoadDlg	WCLD
WinProcessDlg	WCPD
Drawing Management	
Drawing	
WinBeginPaint	WIBPNT
WinEnableWindowUpdate	WIEWU
WinEndPaint	WIEPNT
WinExcludeUpdateRegion	WIEUPR
WinGetClipPS	WPGCPS
WinGetPS	WPGPS
WinGetScreenPS	WPGSPS
WinInvalidateRect	WINVR

C/MASM name	COBOL/FORTRAN name
WinInvalidateRegion	WINVRG
WinIsWindowShowing	WIQSHO
WinIsWindowVisible	WIQVIS
WinLockVisRegions	WILKVR
WinOpenWindowDC	WIOPDC
WinQueryUpdateRect	WIQURC
WinQueryUpdateRegion	WIQURG
WinReleasePS	WPREPS
WinShowWindow	WISHOW
WinUpdateWindow	WIUPDT
WinValidateRect	WIVREC
WinValidateRegion	WIVRGN
Drawing Helpers	
WinDrawBitmap	WIDBMP
WinDrawBorder	WIBORD
WinDrawPointer	WMDRPT
WinDrawText	WIDTXT
WinFillRect	WIFRCT
WinGetSysBitmap	WIQSBM
WinInvertRect	WINVER
WinQueryPresParam	WMQPP
WinRemovePresParam	WMRPP
WinScrollWindow	WISCRL
WinSetPresParam	WMSPP
Dynamic Data Exchange (DDE) Support	
WinDdeInitiate	not used
WinDdePostMsg	not used
WinDdeRespond	not used
Error Processing	
WinCatch	not used
WinFreeErrorInfo	WMFERI
WinGetErrorInfo	WMGEI
WinGetLastError	WMGLE
WinSetErrorInfo	not used
WinThrow	not used
Heap Manager	
WinAllocMem	not used
WinAvailMem	not used
WinCreateHeap	not used
WinDestroyHeap	not used

C/MASM name	COBOL/FORTRAN name
WinFreeMem	not used
WinLockHeap	not used
WinReallocMem	not used
Help Manager (C only)	
WinAssociateHelpInstance	not used
WinCreateHelpInstance	not used
WinCreateHelpTable	not used
WinDestroyHelpInstance	not used
WinLoadHelpTable	not used
WinQueryHelpInstance	not used
Initialization and Termination	
WinCancelShutdown	WMCSD
WinInitialize	WMINIT
WinQueryAnchorBlock	WIQHAB
WinQueryVersion	WMQVER
WinTerminate	WMTERM
Installed Program List	
WinAddProgram	WMADPR
WinCreateGroup	WMCGRP
WinInstStartApp	WMISAP
WinQueryDefinition	WMQDEF
WinQueryProgramTitles	WMQPTI
WinTerminateApp	WMSTOP
Keyboard	
WinEnablePhysInput	WDEPI
WinFocusChange	WMFOCC
WinGetKeyState	WMKEYS
WinGetPhysKeyState	WMGPKS
WinQueryFocus	WMQFOC
WinSetFocus	WMSFOC
WinSetKeyboardStateTable	WMSKST
Library Support	
WinDeleteLibrary	WMDLIB
WinDeleteProcedure	WNDPR
WinLoadLibrary	WMLLIB
WinLoadProcedure	WMLPR
Menus	
WinCreateMenu	not used
WinLoadMenu	WCLM

C/MASM name	COBOL/FORTRAN name
Message Management	
WinBroadcastMsg	WMBRMG
WinCreateMsgQueue	WMCMQ
WinDefAVioWindowProc	not used
WinDestroyMsgQueue	WMDMQ
WinDispatchMsg	WMDISP
WinGetMsg	WMGET
WinInSendMsg	WCISMG
WinMsgMuxSemWait	WMMSWT
WinMsgSemWait	WMSWT
WinPeekMsg	WMPEEK
WinPostMsg	WMPOST
WinPostQueueMsg	WCPQM
WinQueryMsgPos	WCQMPO
WinQueryMsgTime	WCQMTI
WinQueryQueueInfo	WMQQI
WinQueryQueueStatus	WCQSTA
WinRegisterUserDatatype	WMRUD
WinRegisterUserMsg	WMRUM
WinSendDlgItemMsg	WCSDIM
WinSendMsg	WMSEND
WinSetClassMsgInterest	WMSCMI
WinSetMsgInterest	WMMINT
WinSetMsgMode	WMSMM
WinSetSynchroMode	WMSSM
WinWaitMsg	WMWAIT
Mouse Capture	
WinQueryCapture	WMQCAP
WinSetCapture	WMSCAP
Mouse Tracking	
WinShowTrackRect	WISTR
WinTrackRect	WITREC
Initialization File	
WinQueryProfileData	WMQPDA
WinQueryProfileInt	WMQPIN
WinQueryProfileSize	WMQPSZ
WinQueryProfileString	WMQPST
WinWriteProfileData	WMWPDA
WinWriteProfileString	WMWPST

C/MASM name	COBOL/FORTRAN name
Pointer	
WinCreatePointer	WICPOI
WinCreatePointerIndirect	WICPIN
WinDestroyPointer	WMDPOI
WinLoadPointer	WMLPOI
WinQueryPointer	WMQPTR
WinQueryPointerInfo	WMQPTD
WinQueryPointerPos	WMQPTP
WinQuerySysPointer	WMQSPT
WinSetPointer	WMSPTR
WinSetPointerPos	WMSPTP
WinShowPointer	WMSHPT
Rectangles	
WinCopyRect	WICR
WinEqualRect	WIEREC
WinInflateRect	WINFLR
WinIntersectRect	WINTSR
WinIsRectEmpty	WIREMP
WinMakeRect	not used
WinOffsetRect	WIOFFR
WinPtInRect	WIPTRC
WinSetRect	WISREC
WinSetRectEmpty	WISRE
WinSubtractRect	WISUBR
WinUnionRect	WIUNRC
Standard Window	
WinCalcFrameRect	WICRCT
WinCreateFrameControls	WIFRCO
WinCreateStdWindow	WICRTS
String/Character and Code Pages	
WinCompareStrings	WUCOMP
WinCpTranslateChar	WUTRCH
WinCpTranslateString	WUTRST
WinLoadString	WMLSTR
WinNextChar	WUNXTC
WinPrevChar	WUPREC
WinQueryCp	WMGECP
WinQueryCpList	WMQCPL
WinSetCp	WMSECP
WinSubstituteStrings	WUSUBS

C/MASM name	COBOL/FORTRAN name
WinUpper	WUPPER
WinUpperChar	WUPCH
Task List	
WinAddSwitchEntry	WMASWE
WinChangeSwitchEntry	WMCSWE
WinCreateSwitchEntry	WMCRSE
WinQuerySessionTitle	WMQSTI
WinQuerySwitchEntry	WMQSEN
WinQuerySwitchHandle	WMQSHD
WinQuerySwitchList	WMQSLI
WinQueryTaskSizePos	WMQTSP
WinQueryTaskTitle	WMQTTI
WinRemoveSwitchEntry	WMRSWE
WinSwitchToProgram	WMSTP
System and Queue Hooks	
WinCallMsgFilter	WMCMFI
WinReleaseHook	WMRHK
WinSetHook	WMSHOO
System Colors	
WinQuerySysColor	WIQSCO
WinSetSysColors	WISCOL
System Modal Management	
WinQuerySysModalWindow	WIQSMW
WinSetSysModalWindow	WISSMW
System Values	
WinQuerySysValue	WMQSYS
WinSetSysValue	WMSSYS
Timers	
WinGetCurrentTime	WIGCTI
WinStartTimer	WISTAT
WinStopTimer	WISTOT
Window Management	
Activation, Size and Position	
WinGetMinPosition	WIGMNP
WinQueryActiveWindow	WMQACT
WinQueryWindowPos	WIQPOS
WinSetActiveWindow	WMSACT
WinSetMultWindowPos	WISMPS
WinSetWindowPos	WISPOS

C/MASM name	COBOL/FORTRAN name
Creation and Class Information	
WinCreateWindow	WICRT
WinDefWindowProc	WMDWP
WinDestroyWindow	WIDEL
WinQueryClassInfo	WMQCLI
WinQueryClassName	WMQCLN
WinRegisterClass	WMRCL
WinRegisterWindowDestroy	WMREWD
WinSubclassWindow	not used
General Window Information	
WinEnableWindow	WISENA
WinIsThreadActive	WMTHRA
WinIsWindow	WISWIN
WinIsWindowEnabled	WIQENA
WinQueryDesktopWindow	WIQDTW
WinQueryObjectWindow	WIQOBJ
WinQueryWindowDC	WIQWDC
WinQueryWindowProcess	WIQPRO
WinQueryWindowRect	WIQRCT
WinWindowFromDC	WIQHDC
WinWindowFromID	WIQHID
WinWindowFromPoint	WIWFP
Locking	
WinLockWindow	WILKWN
WinLockWindowUpdate	WMLKWU
WinQueryWindowLockCount	WIQWLC
Window Hierarchies	
WinBeginEnumWindows	WIBEWN
WinEndEnumWindows	WIEEW
WinEnumDlgItem	WCEDLI
WinGetNextWindow	WIGWIN
WinIsChild	WICHLD
WinMultWindowFromIDs	WIMWID
WinQueryWindow	WIQWIN
WinSetOwner	WISOWN
WinSetParent	WISPAR
Window Text	
WinQueryDlgItemShort	WCQDIS
WinQueryDlgItemText	WMQDIT
WinQueryDlgItemTextLength	WMQDIL

C/MASM name	COBOL/FORTRAN name
WinQueryWindowText	WIQTXT
WinQueryWindowTextLength	WIQTXL
WinSetDlgItemShort	WCSDI
WinSetDlgItemText	WMSDIT
WinSetWindowText	WISTXT
Window Words	
WinQueryWindowPtr	WIQWPT
WinQueryWindowULong	WIQWUL
WinQueryWindowUShort	WIQWUS
WinSetWindowBits	WISWBT
WinSetWindowPtr	WISWPT
WinSetWindowULong	WISWUL
WinSetWindowUShort	WISWUS

WinAddAtom — Add Atom

This call adds an atom to an atom table.

```
WinAddAtom (AtomTbl, AtomName, atom)
```

Parameters

AtomTbl *(HATOMTBL)* — input
Atom-table handle.

This is the handle returned by a previous WinCreateAtomTable or WinQuerySystemAtomTable call.

AtomName *(STRL)* — input
Atom name.

This is a character string to be added to the table.

If the string begins with a "#" character, the five ASCII digits that follow are converted into an integer atom. If this integer is a valid integer atom, this call returns that atom, without modifying the atom table.

If the string begins with a "!" character, the next two bytes are interpreted as an atom. If it is an integer atom, that atom is returned. If it is not an integer atom and it is a valid atom for the given atom table (that is, it has an atom name and use count associated with it) the use count of that atom is incremented by one and the atom is returned. Otherwise zero is returned.

If the high order word of the string is minus one, the low order word is an atom. If it is an integer atom, that atom is returned. If it is not an integer atom and it is a valid atom for the given atom table (that is, it has an atom name and use count associated with it) the use count of that atom is incremented by one and the atom is returned. Otherwise zero is returned.

atom *(ATOM)* — return
Atom value:

Atom The atom associated with the passed string
0 Invalid atom-table handle or invalid atom name specified.

Possible returns from WinGetLastError:

PMERR_INVALID_HATOMTBL
PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND

Remarks

If the atom name represents an integer atom, this call returns the atom represented by the passed atom name.

If the atom name does not represent an integer atom and if the atom name already exists in the atom table, this call increments the use count of that atom by one. Otherwise, the atom is added to the table and its use count set to one. In either case this call returns the atom represented by the passed atom name.

WinAddProgram — Add Program

This call adds a program entry to the installed program list.

WinAddProgram (hab, ProgramInfo, GroupHandle, *ProgHandle*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

ProgramInfo *(PIBSTRUCT)* — input
Program information for the program to be added to the installed program list.

The initial window position and size information contained within this parameter is obtained from the initialization file. If both of these fields are set to zero, the system generates a suggested starting size and position.

The visibility attribute controls whether the group or program is visible in the startup list (and therefore can be started by the user).

GroupHandle *(HPROGRAM)* — input
Handle of the program group to which the program is to be added.

The group must not have the protected attribute.

The SGH_ROOT program group handle is reserved for the user interface shell.

NULL Add to the default group (the first defined group)
Other Program-group handle.

ProgHandle *(HPROGRAM)* — return
Program handle for the program added to the installed program list.

NULL Error occurred
Other Program handle.

Possible returns from WinGetLastError:

PMERR_INVALID_GROUP_HANDLE
PMERR_INVALID_TITLE
PMERR_NOT_IN_IDX
PMERR_NOT_CURRENT_PL_VERSION
PMERR_INSUFF_SPACE_TO_ADD
PMERR_INVALID_TARGET_HANDLE
PMERR_DUPLICATE_TITLE
PMERR_MEMORY_DEALLOCATION_ERR

Remarks

Program titles need not be unique, although duplicate titles within the same group are not allowed.

See also the PrfAddProgram call, which should be used in preference to this one. PrfAddProgram must be used to add a program whose executable has a long file name, since WinAddProgram uses a *PIBSTRUCT*, which has a fixed size character array to hold the executable file name (see the **executable** field in *PIBSTRUCT*).

This call requires the existence of a message queue.

WinAddSwitchEntry — Add Switch Entry

This call adds an entry to the Task list.

```
WinAddSwitchEntry (SwitchData, Switch)
```

Parameters

SwitchData *(SWCNTRL)* — input
Switch data.

Contains information about the newly-created task list entry.

If the **swtitle** field of the *SWCNTRL* structure is NULL, the system uses the name under which the application is started. This only applies for programs written for OS/2 Versions 1.1 and later, and only for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

Leading and trailing blanks are removed from the title, and if necessary it is truncated to 60 characters.

If the **hprog** field of the *SWCNTRL* structure is NULL, the value used by the system when the program was loaded (if it has been loaded) is substituted.

If the **process** field of the *SWCNTRL* structure is NULL, the current process ID is used.

If the **session** field of the *SWCNTRL* structure is NULL, the current session ID is used.

If the **icon** field of the *SWCNTRL* structure is NULL, the system supplies a default icon.

Switch *(HSWITCH)* — return
Handle to the newly-created task list entry.

There is a system limit to the number of task list entries. However, this is a large number (several hundred) and is unlikely to be reached in practice since other system limits, such as memory size, restrict the number before this limit is reached.

NULL Error occurred
Other Handle to the newly-created task list entry.

Possible returns from WinGetLastError:

PMERR_NO_SPACE
PMERR_INVALID_WINDOW
PMERR_INVALID_SESSION_ID

Remarks

Both this call and the WinRemoveSwitchEntry call are not required if the main window is created with the frame creation flags FCF_TASKLIST or FCF_STANDARD, as these flags automatically update the task list when the main window is created or destroyed.

This call requires the existence of a message queue.

WinAlarm — Sound Alarm

This call generates an audible alarm.

WinAlarm (DeskTop, Style, *Result*)

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop window
Other	Specified desktop window.

Style *(BIT16)* — input
Alarm style.

Used to signify different situations to the operator.

The duration and frequency of the alarms can be changed by the WinSetSysValue call. The alarm frequency is defined to be in the range X'0025' through X'7FFF'. The alarm is not generated if system value SV_ALARM is set to FALSE. The alarms are dependent on the device capability.

Different alarms are selected by use of these values:

WA_WARNING
WA_NOTE
WA_ERROR

Result *(BOOL)* — return
Alarm-generated indicator:

TRUE	Alarm generated
FALSE	Alarm not generated.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG

Remarks

Although this call is in the INCL_WINDIALOGS section, it is part of the common subset and is also included if INCL_COMMON is defined.

WinAllocMem – Allocate Heap Space

This call allocates memory from a heap.

WinAllocMem (Heap, cb, *Mem*)

Parameters

Heap *(HHEAP)* – input
Heap handle.

Handle to a heap, returned by a previous call to WinCreateHeap.

cb *(USHORT)* – input
Total number of bytes to allocate.

Mem *(SEGOFF)* – return
Memory pointer.

The short pointer is relative to the beginning of the segment that contains the heap:

NULL This call is unable to allocate the memory object, either because an invalid heap handle is specified, or there is not enough room in the heap for an object of the specified size and it is unable to grow the segment containing the heap by an amount large enough to satisfy the request.

Other Near pointer to the allocated memory block.

Remarks

This call 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.

If the passed heap is created with the HM_MOVEABLE option, then the value of the **cb** parameter is remembered in the second reserved word of the allocated block. The returned address is then the address of the first reserved word.

The allocation algorithm first searches the dedicated free lists, starting with the one whose size is greater than, or equal to, the requested size. It searches the dedicated free lists, until either it finds the smallest block greater than, or equal to, 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. (This may not be the smallest free block that would satisfy the request, because 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 call next attempts to obtain space by calling WinAvailMem. 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 minimum growth parameter specified on the WinCreateHeap call. If that fails, then this call returns NULL.

WinAssociateHelpInstance — Associate Help Instance

This call associates the specified instance of the help manager with the window chain of the specified application window.

WinAssociateHelpInstance (HelpInstance, App, *Success*)

Parameters

HelpInstance *(HWND)* — input
Handle of an instance of the help manager.

This is the handle returned by the WinCreateHelpInstance call.

NULL Disassociate an instance of the help manager from a window chain when the instance has been destroyed.
Other The handle of an instance of the help manager to be associated with the application window chain.

App *(HWND)* — input
Handle of an application window.

The handle of the application window with which the instance of the help manager will be associated. The instance of the help manager is associated with the application window and any of its children and owned windows.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

In order to provide help, the application must associate an instance of the help manager with a chain of application windows. This association lets the help manager know which instance should provide the help function.

The help manager traces the window chain, starting from the window where help is requested. The application window in the chain with the associated help instance will be the one with which the help manager communicates and positions the help window next to, unless a HM_SET_ACTIVE_WINDOW message is sent to the help manager. If the HM_SET_ACTIVE_WINDOW message is sent to the help manager, the active window parameter is the window with which the help manager communicates. The help manager positions the help window next to the window specified as the relative window.

This call requires the existence of a message queue.

WinAvailMem – Query Available Heap Space

This call returns the size of the largest free block on the heap.

WinAvailMem (Heap, Compact, b, *Largest*)

Parameters

Heap *(HHEAP)* – input
Heap handle.

Handle to a heap, returned by a previous WinCreateHeap call.

Compact *(BOOL)* – input
Compact indicator:

TRUE Reorganize the heap.

If the passed heap is created with the HM_MOVEABLE option, the reorganization consists of moving all moveable blocks towards the beginning of the heap. The low order bit of the handle word is used to identify allocated blocks when scanning the heap. If it is zero, then it is free memory and it is overwritten. The presence of fixed objects may inhibit the amount of motion that can occur. While the compaction is occurring the dedicated and non-dedicated free lists are reconstructed from any free blocks that cannot be filled by the compactor.

If the passed heap is not created with the HM_MOVEABLE option, the reorganization consists of sorting all the free lists into a single list in address order; scanning the list looking for adjacent blocks to coalesce; and finally reconstructing the dedicated free lists and the non-dedicated free list.

FALSE Do not reorganize the heap.

b *(COUNT2)* – input
Reserved.

NULL Reserved value.

Largest *(USHORT)* – return
Size of the largest memory block available.

65535 Error occurred
Other Size of the largest memory block available.

Possible returns from WinGetLastError:

PMERR_INVALID_HHEAP
PMERR_ATOM_NAME_NOT_FOUND
PMERR_INVALID_HEAP_POINTER
PMERR_INVALID_HEAP_SIZE_WORD
PMERR_INVALID_HEAP_SIZE_PARM
PMERR_HEAP_MAX_SIZE_REACHED
PMERR_HEAP_OUT_OF_MEMORY

Remarks

This call is expensive in terms of time.

WinBeginEnumWindows — Begin Window Enumeration

This call begins the enumeration process for all of the immediate child windows of a specified window.

WinBeginEnumWindows (Parent, *Henum*)

Parameters

Parent *(HWND)* — input
Handle of the window whose child windows are to be enumerated:

HWND_DESKTOP	Enumerate all main windows
HWND_OBJECT	Enumerate all object windows
Other	Enumerate all immediate children of the specified window.

Henum *(HENUM)* — return
Enumeration handle.

This is used in subsequent calls to the WinGetNextWindow call to return the immediate child-window handles in succession.

When the application has finished the enumeration, the enumeration handle must be destroyed with the WinEndEnumWindows call.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call remembers the window hierarchy at the time of invocation of the call. Thereafter the information is referenced by use of the **Henum** parameter and does not change during the enumeration by the WinGetNextWindow call. The windows are enumerated in the z-order at the time enumeration is begun, with the topmost child window enumerated first. This ensures that all the child windows are enumerated.

Only the immediate children of the specified window are enumerated, child windows of the child windows are excluded.

The enumerated windows are not locked by this call and can thus be destroyed between the time that it is called and the time that the WinGetNextWindow call is used to obtain the handle for the window.

WinBeginPaint — Begin Paint

SAA

This call obtains a presentation space whose associated update region is set ready for drawing in a specified window.

WinBeginPaint (hwnd, hps, Rect, *PaintPS*)

Parameters

hwnd *(HWND)* — input
Handle of window where drawing is going to occur:

HWND_DESKTOP	The desk top window
Other	Specified window.

hps *(HPS)* — input
Presentation-space handle:

NULL	Obtain a cache presentation space.
Other	Presentation-space handle. This call sets its clipping region to the update region of the **hwnd** parameter.

Rect *(RECT)* — output
Bounding rectangle:

Programming Note: The data type *WRECT* may also be used, if supported by the language.

NULL	No bounding rectangle; that is, repainting is not required.
Other	Specifies the smallest rectangle bounding the update region, in window coordinates.

PaintPS *(HPS)* — return
Presentation-space handle.

Handle of presentation space whose associated update region is set for drawing in the window.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is generally made during the processing of a WM_PAINT message when the application needs to update the content of the window. If the presentation space already exists, its update region is set and the device context of the window is associated with the presentation space. Otherwise, a cache presentation space is obtained specifically for the window.

The update region associated with **hwnd** is reset to NULL. It is assumed that any drawing following this call restores the content of the window to a fully correct state.

This call hides the pointer if it is in the window and the WinEndPaint call restores it. This call also hides the tracking rectangle if it is active and might hide part of the painting window; that is, if **hwnd** is a child of the window specified by the **hwnd** parameter in the WinTrackRect call. The WinEndPaint call shows it again. WinEndPaint must be called after the application completes drawing, and may be nested for the same window.

This call requires the existence of a message queue.

WinBroadcastMsg — Broadcast Message

This call broadcasts a message to multiple windows.

WinBroadcastMsg (Parent, Msgid, Param1, Param2, Cmd, *Success***)**

Parameters

Parent *(HWND)* — input
Parent-window handle.

Msgid *(USHORT)* — input
Message identifier.

Param1 *(MPARAM)* — input
Parameter 1.

Param2 *(MPARAM)* — input
Parameter 2.

Cmd *(BIT16)* — input
Broadcast message command:

BMSG_POST	Post the message. This value is mutually exclusive with BMSG_SEND and BMSG_POSTQUEUE.
BMSG_SEND	Send the message. This value is mutually exclusive with BMSG_POST and BMSG_POSTQUEUE.
BMSG_POSTQUEUE	Post a message to all threads that have a message queue. This value is mutually exclusive with BMSG_POST and BMSG_SEND. The **hwnd** parameter of the *QMSG* structure is set to NULL.
BMSG_DESCENDANTS	Broadcast the message to all the descendants of the **Parent** parameter.
BMSG_FRAMEONLY	Broadcast the message only to windows with a style of CS_FRAME.

Success *(BOOL)* — return
Success indicator:

TRUE	Message was sent or posted successfully to all applicable windows
FALSE	Error occurred.

Remarks

This call sends or posts a message to all the immediate child windows of **Parent**, except in the case when **Cmd** is BMSG_DESCENDANTS.

The **Msgid**, **Param1**, and **Param2** parameters make up the message sent or posted. The window handle of the receiving window is added to the message.

WinCalcFrameRect — Calculate Frame Rectangle

SAA

This call calculates a client rectangle from a frame rectangle, or a frame rectangle from a client rectangle.

WinCalcFrameRect (hwnd, Rect, Frame, *Success***)**

Parameters

hwnd *(HWND)* — input
Frame-window handle.

Rect *(RECT)* — input/output
Window rectangle.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

Frame *(BOOL)* — input
Frame indicator:

TRUE Frame rectangle provided
FALSE Client-area rectangle provided.

Success *(BOOL)* — return
Rectangle-calculated indicator:

TRUE Rectangle successfully calculated
FALSE Error occurred, or the calculated rectangle is empty.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call provides the size and position of the client area within the specified frame rectangle for the specified frame window, or conversely, the size and position of the frame window that would contain a client window of the specified size and position.

This call sends a WM_CALCFRAMERECT message to the frame window. This enables a subclassed frame control to correctly implement the calculation.

This call works if window **hwnd** is hidden; window **hwnd** should be hidden if it is required that the window shows a particular client rectangle when the window is first shown.

This call requires the existence of a message queue.

WinCallMsgFilter — Call Message Filter

This call calls a message-filter hook.

WinCallMsgFilter (hab, pqmsg, Filter, *HookRet*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

pqmsg *(QMSG)* — input
Message to be passed to the message-filter hook.

Filter *(USHORT)* — input
Filter.

Message-filter code passed to the message-filter hook. This can be one of the standard MSGF_* values (see MsgFilterHook) or an application-specific value.

HookRet *(BOOL)* — return
Message-filter hook return indicator:

TRUE A message-filter hook returns TRUE
FALSE All message-filter hooks return FALSE, or no message-filter hooks are defined.

Remarks

This call allows an application to pass a message to the message-filter hook procedure(s).

This call requires the existence of a message queue.

WinCancelShutdown — Cancel Shutdown

This call cancels a request for an application to shut down.

```
WinCancelShutdown (hmq, CancelAlways, Success)
```

Parameters

hmq *(HMQ)* — input
Queue handle.

CancelAlways *(BOOL)* — input
Cancelation control.

TRUE No WM_QUIT message should be placed on this queue during system shutdown.
FALSE The application wants to ignore any outstanding WM_QUIT messages already sent to it, but a message should be sent during other system shutdowns.

Success *(BOOL)* — return
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Remarks

This call is used to respond to a WM_QUIT message or to stop further WM_QUIT messages being sent to this queue by the system.

This call requires the existence of a message queue.

WinCatch – Save Execution Environment

This function saves the execution environment.

WinCatch (CatchBuf, *RetCode***)**

Parameters

CatchBuf *(CATCHBUF)* – output
Saved execution environment buffer.

The state of all system registers and the instruction counter are saved.

RetCode *(SHORT)* – return
Return code.

0 Execution environment is saved.
Other Execution environment is restored and this value is that provided by the complementary WinThrow function.

Remarks

The execution environment is saved in the **CatchBuf** parameter and the **RetCode** parameter is always returned as 0 when the execution environment is saved.

The complementary WinThrow function restores the execution environment from a previously saved **CatchBuf** parameter and provides a value which becomes the value of the **RetCode** parameter. Therefore the code immediately following this function can analyze this return value and perform appropriate processing.

Programming Note: This technique is particularly useful for handling error conditions at a single point in the structure of an application.

This is achieved by using the WinCatch function at the desired point in the application at which error conditions are to be handled. On the invocation of this function the execution environment is saved and a 0 value is returned, on which the application proceeds normally. If subsequently some other part of the application detects an error condition, it can use the WinThrow function with an appropriate return value. This causes control to pass to the point in the application at which the error condition is to be handled, but on this occasion with the return code provided by the part of the application which detected the error condition. Then, the error condition handling portion of the application can take appropriate action based on the value of the return code.

Note that any resources that have been allocated between the issue of this function and the complementary WinThrow function must be released by the error condition handling portion of the application.

This call requires the existence of a message queue.

WinChangeSwitchEntry — Change Switch Entry

This call changes the information in a task-list entry.

```
WinChangeSwitchEntry (Switch, SwitchData, RetCode)
```

Parameters

Switch *(HSWITCH)* — input
Handle to the task-list entry to be changed.

SwitchData *(SWCNTRL)* — input
Switch-control data.

Contains the information to change the task-list entry.

RetCode *(USHORT)* — return
Return code:

0	Successful completion
Other	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_SWITCH_HANDLE
PMERR_INVALID_WINDOW

Remarks

Leading and trailing blanks are removed from the title and, if necessary, it is truncated to 60 characters.

This call requires the existence of a message queue.

WinCloseClipbrd — Close Clipboard

This call closes the clipboard, allowing other applications to open it for use.

```
WinCloseClipbrd (hab, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call causes the contents of the clipboard to be drawn in the clipboard viewer window (if any), by sending it a WM_DRAWCLIPBOARD message.

The clipboard must be open before this call is invoked.

This call requires the existence of a message queue.

WinCompareStrings — Compare Strings

Compares two null-terminated strings defined using the same code page.

```
WinCompareStrings (hab, Codepage, CountryCode, String1, String2, Options, Result)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Codepage *(IDENTITY)* — input
Code page identity of both strings.

CountryCode *(IDENTITY)* — input
Country code.

String1 *(STRL)* — input
String 1.

String2 *(STRL)* — input
String 2.

Options *(BIT16)* — input
Reserved.

NULL Reserved value.

Result *(USHORT)* — return
Comparison result.

WCS_EQ	Strings are equal
WCS_LT	String 1 is less than string 2
WCS_GT	String 1 is greater than string 2
WCS_ERROR	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_STRING_PARM

Remarks

This call requires the existence of a message queue.

WinCopyAccelTable — Copy Accelerator Table

This call is used to get the accelerator-table data corresponding to an accelerator-table handle, or to determine the size of the accelerator-table data.

WinCopyAccelTable (hAccel, AccelTable, CopyMax, *Copied***)**

Parameters

hAccel *(HACCEL)* — input
Accelerator-table handle.

AccelTable *(ACCELTABLE)* — input/output
Accelerator-table data area:

NULL Return the size, in bytes, of the complete accelerator table, and ignore the **CopyMax** parameter.
Other Copy up to **CopyMax** bytes of the accelerator table into this data area.

CopyMax *(USHORT)* — input
Maximum data area size.

Copied *(USHORT)* — return
Amount copied or size required:

Other Amount of data copied into the data area, or the size of data area required for the complete accelerator table.
0 Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HACCEL

Remarks

This call requires the existence of a message queue.

WinCopyRect – Copy Rectangle

This call copies a rectangle.

WinCopyRect (hab, Dest, Src, *Success*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

Dest *(RECT)* – output
Destination rectangle.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

Src *(RECT)* – input
Source rectangle.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

WinCpTranslateChar – Translate Character with Code Page

This call translates a character from one code page to another.

WinCpTranslateChar (hab, CpSource, Source, CpDest, *Dest*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

CpSource *(IDENTITY)* – input
Source-character code page.

Source *(UCHAR)* – input
Character to be translated.

CpDest *(IDENTITY)* – input
Code page of the resultant character.

Dest *(UCHAR)* – return
If nonzero, the translated character.

If the source character cannot be translated to the selected code page, this call returns the character X'FF'.

Possible returns from WinGetLastError:

PMERR_INVALID_STRING_PARM
PMERR_INVALID_SRC_CODEPAGE
PMERR_INVALID_DST_CODEPAGE

Remarks

Successful code page translation can only take place if there is a code point in the target code page that corresponds to the code point in the source code page.

This call requires the existence of a message queue.

WinCpTranslateString – Translate String with Code Page

This call translates a string from one code page to another.

WinCpTranslateString (hab, CpSource, Source, CpDest, LenDest, Dest, *Success***)**

Parameters

hab *(HAB)* – input
Anchor-block handle.

CpSource *(IDENTITY)* – input
Source-string code page.

Source *(STRL)* – input
String to be translated.

This is a null-terminated string.

CpDest *(IDENTITY)* – input
Code page of the resultant string.

LenDest *(COUNT2B)* – input
Maximum length of output string.

An error is raised if this is not large enough to contain the translated string.

Dest *(STRL)* – output
The translated string.

This is a null-terminated string.

If a source character cannot be translated to the selected code page, this call substitutes the character X'FF', and returns FALSE.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_STRING_PARM
PMERR_INVALID_SRC_CODEPAGE
PMERR_INVALID_DST_CODEPAGE
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

Successful code page translation can only take place if there is a code point in the target code page that corresponds to the code point in the source code page.

This call requires the existence of a message queue.

WinCreateAccelTable — Create Accelerator Table

This call creates an accelerator table from the accelerator definitions in memory.

WinCreateAccelTable (hab, AccelTable, *hAccel***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

AccelTable *(ACCELTABLE)* — input
Accelerator table.

hAccel *(HACCEL)* — return
Accelerator-table handle.

Remarks

This call returns a different **hAccel** value when called twice in succession with the same parameter values.

This call requires the existence of a message queue.

WinCreateAtomTable — Create Atom Table

This call creates an empty atom table of the specified size.

```
WinCreateAtomTable (Initial, Buckets, AtomTbl)
```

Parameters

Initial *(USHORT)* — input
Initial bytes.

Initial number of bytes to be reserved for the atom table. This is a lower bound on the amount of memory reserved. The amount of memory actually used by an atom table depends upon the actual number of atoms stored in the table. If zero, the size of the atom table is the minimum size needed to store the atom hash table.

Buckets *(USHORT)* — input
Size of the hash table.

Used to access atoms. If zero, the default value of 37 is used. The best results are achieved if a prime number is used.

AtomTbl *(HATOMTBL)* — return
Atom-table handle:

NULL Call failed.
Other Atom-table handle. This must be passed as a parameter in subsequent atom manager calls.

Remarks

The minimum size of atom table allocated is 16+(2*__Buckets__).

The atom table is owned by the process from which this call is issued. It cannot be accessed directly from any other process. If it is still in existence when the process terminates, it will automatically be deleted by the system.

There is a system atom table which is created at boot time, which cannot be destroyed, and which can be accessed by any process in the system. The handle of the system atom table is queried with the WinQuerySystemAtomTable call.

WinCreateCursor — Create Cursor

This call creates or changes a cursor for a specified window.

WinCreateCursor (hwnd, x, y, cx, cy, rgf, Clip, *Success*)

Parameters

hwnd *(HWND)* — input
Handle of window in which cursor is displayed.

This must be the handle of a window for which the application can receive input.

x *(SHORT)* — input
x position of cursor.

y *(SHORT)* — input
y position of cursor.

cx *(SHORT)* — input
x size of cursor.

If 0, the system nominal border width (SV_CXBORDER) is used.

cy *(SHORT)* — input
y size of cursor.

If 0, the system nominal border height (SV_CYBORDER) is used.

rgf *(USHORT)* — input
Controls the appearance of the cursor:

CURSOR_SOLID	The cursor is solid.
CURSOR_HALFTONE	The cursor is halftone.
CURSOR_FRAME	The cursor is a rectangular frame.
CURSOR_FLASH	The cursor flashes.
CURSOR_SETPOS	Set a new cursor position. **cx** and **cy** are ignored. Used when a cursor has already been created. In this case, all other appearance flags are ignored.

Clip *(RECT)* — input
Cursor rectangle.

A rectangle within which the cursor is visible. If the cursor goes outside this rectangle, it is clipped away and is invisible.

The rectangle is specified in window coordinates.

If **Clip** is NULL, the drawing of the cursor is clipped to the window rectangle of **hwnd**.

Programming Note: The cursor is always clipped to the window rectangle, even if part of **Clip** is outside it.

The data type *WRECT* can also be used, if supported by the language.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG

WinCreateCursor — Create Cursor

SAA

Remarks

The cursor is used to indicate the position of text input. It is initially hidden and must be made visible using the WinShowCursor call.

This call destroys any existing cursor, as it is confusing to the user for there to be two cursors visible at any one time. An application creates and displays a cursor when it has the input focus, or is the active window. Creating a cursor at any other time can stop the cursor from flashing in another window. Similarly, when the application loses the input focus or becomes inactive, it destroys its cursor using the WinDestroyCursor call.

The cursor width is generally specified as 0 (nominal border width is used). This is preferable to a value of 1, for example, as such a fine width is almost invisible on a high-resolution device.

This call requires the existence of a message queue.

WinCreateDataStructure — Create Data Structure

This call creates a data structure in a standard form and returns a handle for it.

WinCreateDataStructure (hab, Count, Types, InLen, InStruc, MaxLen, Handle, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Count *(USHORT)* — input
Number of elements.

This is the number of elements in the **Types** array. It must be at least one.

If the **Types** array contains a data type that does not have an implied length, (BUFFER, STR, and STRL) **Count** must have a value of 1.

Types *(SHORT*Count)* — input
Data type codes.

These define the layout of the input structure **InStruc**. They may be simple data types, structure data types, pointer data types, control data types, or any valid combination.

Note that not all of the data types that occur can be specified on this call.

InLen *(USHORT)* — input
Length of input structure.

This is the length in bytes of the input structure **InStruc**. It must not be less than the length implied by the definition of the structure contained in the **Types** array. For those data types that do not have an implied length, the value of **InLen** determines how much of **InStruc** is used.

InStruc *(BUFFER)* — input
Input structure.

This is the structure in application form, from which the standard form is to be created. The application form of the structure depends on the programming language of the application.

MaxLen *(USHORT)* — input
Maximum length of structure.

This is the length in bytes of the storage that is to be allocated to contain the standard form of the data structure.

If zero, the storage allocated must be big enough to accommodate the standard form of the structure.

If greater than zero, it must be at least as big as the size of the standard form implied by the **Types** array. A value of **MaxLen**, which is one byte bigger than **InLen** will accommodate most structures.

The WinModifyDataStructure call may be used later to increase the size of the data structure, if it does not cause the standard form of the structure to exceed the size specified by this parameter.

Handle *(HSTRUCT)* — output
Handle of data structure created.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

WinCreateDataStructure — Create Data Structure

SAA

Remarks

A data structure created with one set of data types may be queried or used with a different set of data types, provided that such use is consistent with the nature of the data in the structure.

The WinDestroyDataStructure call must be used to destroy the data structure when it is no longer required.

Note: There is no requirement for this function in C.

This call requires the existence of a message queue.

WinCreateDlg — Create Dialog

This call creates a dialog window.

WinCreateDlg (Parent, Owner, DlgProc, DlgTmp, CreateParams, *Dlg*)

Parameters

Parent *(HWND)* — input
Parent-window handle of the created dialog window:

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

Owner *(HWND)* — input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg call.

DlgProc *(WNDPROC)* — input
Dialog procedure for the created dialog window.

DlgTmp *(DLGTEMPLATE)* — input
Dialog template.

CreateParams *(CREATEPARAMS)* — input
Application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

Dlg *(HWND)* — return
Dialog-window handle:

NULL	Dialog window not created
Other	Dialog-window handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND

Remarks

This call is identical to the WinLoadDlg call except that it creates a dialog window from **DlgTmp** in memory, rather than a dialog template in a resource file.

This call should not be used while the pointing device capture is set (see WinSetCapture).

This call requires the existence of a message queue.

WinCreateFrameControls — Create Frame Controls

SAA

This call creates the standard frame controls for a specified window.

```
WinCreateFrameControls (Frame, Fcdata, Title, Success)
```

Parameters

Frame *(HWND)* — input
Frame-window handle.

Becomes the parent and owner window of all the frame controls that are created:

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

Fcdata *(FRAMECDATA)* — input
Frame-control data.

This includes a combination of frame creation flags (FCF_*), that specifies which frame controls are to be created. For further information, see "Standard Frame Styles and Frame Creation Flags" on page 15-2.

Title *(STRL)* — input
Title string.

A string that is displayed in the WC_TITLEBAR control when FCF_TITLEBAR is specified in **Fcdata**.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG

Remarks

This call is typically used when the standard frame controls are needed for use with a nonstandard window (such as a non-WC_FRAME class).

All of the controls are created with the standard FID_* window identifiers; see page 15-1.

The controls are created but not formatted. Formatting must be done by setting the required positions. All controls are created with position and size set to zero and WS_VISIBLE is not set.

This call requires the existence of a message queue.

WinCreateGroup — Create Group

This call creates a new program group entry in the installed program list.

WinCreateGroup (hab, Title, Visibility, res1, res2, *GroupHandle***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Title *(STRL)* — input
Title of the new group.

This is displayed to the end-user and so should be readable and meaningful.

The maximum string size is 60 characters. Strings that exceed this limit are truncated to 60 characters. Leading and trailing blanks are removed.

The string must contain at least one nonblank character and must not contain "\".

Visibility *(UCHAR)* — input
Visibility control.

Controls the visibility and other attributes of the new group:

SHE_VISIBLE	Group is visible (can be viewed by end-user)
SHE_INVISIBLE	Group is invisible
SHE_UNPROTECTED	Group is not protected (default)
SHE_PROTECTED	Group is protected.

This implies that a program within the group cannot be changed by use of the PrfChangeProgram function, neither can a program by added to or deleted from the group by the use of the PrfAddProgram, the WinAddProgram or the PrfRemoveProgram functions, respectively. Also, the group cannot be destroyed by use of the PrfDestroyGroup function.

A protected group is changed to an unprotected group by use of the PrfChangeProgram function.

res1 *(BIT32)* — input
Reserved.

NULL Reserved value.

res2 *(BIT32)* — input
Reserved.

NULL Reserved value.

GroupHandle *(HPROGRAM)* — return
Program-group handle for the newly-created group.

NULL Error occurred
Other Program-group handle.

WinCreateGroup — Create Group

Possible returns from WinGetLastError:

PMERR_INVALID_GROUP_HANDLE
PMERR_INVALID_TITLE
PMERR_NOT_IN_IDX
PMERR_NOT_CURRENT_PL_VERSION
PMERR_INSUFF_SPACE_TO_ADD
PMERR_INVALID_TARGET_HANDLE
PMERR_MEMORY_DEALLOCATION_ERR

Remarks

Because the new group is created empty, either the PrfAddProgram or the WinAddProgram call must be used to add program entries to the group.

If the group already exists, the existing group handle is returned and no new group handle is created. (See also the PrfCreateGroup call, which should be used in preference to this one.)

This call requires the existence of a message queue.

WinCreateHeap — Create Heap

This call creates a heap that can be used for memory management.

WinCreateHeap (HeapBase, HeapSize, Grow, MinDed, MaxDed, Options, *hHeap*)

Parameters

HeapBase *(USHORT)* — input
Selector of the segment to contain the local heap.

HeapSize *(USHORT)* — input
Initial heap size in bytes.

Grow *(USHORT)* — input
Heap growth size in bytes.

If the heap has to be increased to satisfy a request for memory from WinAllocMem or WinReallocMem, this determines the minimum number of bytes by which the segment containing the heap must grow.

Zero Heap growth size is 512 bytes
Other Heap growth size.

MinDed *(USHORT)* — input
Minimum element size.

For a dedicated free list. The value of this parameter must be less than or equal to the value of the **MaxDed** parameter. See also the **MaxDed** parameter.

MaxDed *(USHORT)* — input
Maximum element size.

For a dedicated free list. The value of this parameter must be equal to or greater than the value of the **MinDed** parameter.

This parameter and the **MinDed** parameter determine the number of dedicated free lists for the heap. There is one dedicated free list created for each element size (in multiples of four bytes) to cover the range of element sizes specified by the values of these parameters, except that a dedicated free list is not created for an element size of zero. If the value of both these parameters is zero, then no element size has a dedicated free list and all sizes are on the non-dedicated free list.

The cost of each dedicated free list is an additional two bytes in the heap control block.

Options *(BIT16)* — input
Optional characteristics.

HM_MOVEABLE Specifies that the created heap should support moveable objects. This causes two words to be added at the beginning of each allocated object.
HM_VALIDSIZE This option, in conjunction with HM_MOVEABLE, checks the size parameter of WinFreeMem.

hHeap *(HHEAP)* — return
Heap handle.

This heap handle must be passed as the first parameter in subsequent heap calls:

0 Error occurred. No heap is created, either because of lack of memory in the last case described, or an invalid selector in the second and third cases, or no **HeapSize** parameter in the first case.
Other Heap handle.

WinCreateHeap — Create Heap

Possible returns from WinGetLastError:

PMERR_INVALID_FLAG
PMERR_INVALID_SELECTOR

Remarks

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:

- An application's automatic data segment.
- A dynlink package's automatic data segment.
- A segment allocated with DosAllocSeg (public or shared).

To accommodate these various targets for heaps, all four possible combinations of the **HeapBase** and the **HeapSize** parameters are used to distinguish between the various options, as follows:

Heapbase	Heapsize	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 call 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 loader already reserved the space at the end of the data segment, after the static data was loaded from the .EXE file.
selector	not 0	Caller is a dynlink package that wants a heap placed at the end of its automatic data segment. The **HeapSize** 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 call 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	not 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 call DosAllocSeg. See the WinLockHeap call for accessing the base of the segment allocated by WinCreateHeap when called with this combination of parameters.

Elements that have a dedicated free list (see the **MinDed** and **MaxDed** parameters) incur minimal overhead on an allocation. Allocation requests that cannot be satisfied with a free block from a dedicated free list do a linear search of the non-dedicated free list until the first free block large enough to satisfy the request is found.

Note that this function does not detect duplicate calls with the same parameters. Memory will be reallocated if this occurs, without regard to the previous call.

Programming Note: If a Presentation Manager application uses WinCreateHeap, in addition to other OS/2 or C-language memory allocation functions, care must be taken that the calls do not conflict. To ensure this, the application must create a new segment using DosAllocSeg or DosAllocShrSeg, and pass that segment's selector to this call in **HeapBase**. This creates the heap in a different segment.

WinCreateHelpInstance — Create Help Instance

This call creates an instance of the help manager with which to request help manager functions.

WinCreateHelpInstance (hab, HMInitStructure, *help*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

The handle of the application anchor block returned from the WinInitialize call.

HMInitStructure *(HELPINIT)* — input/output
Help manager initialization structure.

help *(HWND)* — return
Help-manager handle.

NULL Error occurred
Other Help-manager handle.

Remarks

If an error occurred, it is in the **returncode** parameter of the *HELPINIT* structure.

This call requires the existence of a message queue.

WinCreateHelpTable — Create Help Table

This call is used to identify or change the help table.

```
WinCreateHelpTable (HelpInstance, HelpTable, Success)
```

Parameters

HelpInstance *(HWND)* — input
Handle of an instance of the help manager.

This is the handle returned by the WinCreateHelpInstance call.

HelpTable *(HELPTABLE)* — input
Help table allocated by the application.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call corresponds to the HM_CREATE_HELP_TABLE message that identifies a help table that is in memory.

This call requires the existence of a message queue.

WinCreateMenu — Create Menu

This call creates a menu window from the menu template.

WinCreateMenu (Owner, Menutmp, *Menu***)**

Parameters

Owner *(HWND)* — input
Owner- and parent-window handle of the created menu window.

If this is HWND_OBJECT or a window handle returned by the WinQueryObjectWindow call, the menu window is created as an object window.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

Menutmp *(MT)* — input
Menu template in binary format.

Menu *(HWND)* — return
Menu-window handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The menu window is created with an identity of FID_MENU.

When a WC_MENU window is created with the WinCreateWindow call, **CtlData** is assumed to be a menu template, which is used to create the menu. If **CtlData** is NULL, an empty menu is created.

This call requires the existence of a message queue.

WinCreateMsgQueue — Create Message Queue

SAA

This call creates a message queue.

WinCreateMsgQueue (hab, Queuesize, *hmq*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Queuesize *(SHORT)* — input
Maximum queue size.

This is the maximum number of messages that can be queued.

0 Use the system default queue size
Other Maximum queue size.

hmq *(HMQ)* — return
Message-queue handle:

NULL Queue cannot be created
Other Message-queue handle.

Possible returns from WinGetLastError:

PMERR_HEAP_MAX_SIZE_REACHED
PMERR_HEAP_OUT_OF_MEMORY
PMERR_RESOURCE_NOT_FOUND

Remarks

Most Presentation Manager calls require a message queue. The WinCreateMsgQueue function call must be issued after the WinInitialize call, but before any other Presentation Manager calls are invoked. It must be issued only once per thread.

WinCreatePointer — Create Pointer

This call creates a pointer from a bit map.

WinCreatePointer (DeskTop, BitMap, PointerSize, xHotspot, yHotspot, *hptr*)

Parameters

DeskTop *(HWND)* — input
Desktop-window handle or HWND_DESKTOP.

BitMap *(HBITMAP)* — input
Bit-map handle from which the pointer image is created.

The bit map must be logically divided into two sections vertically, each half representing one of the two images used as the successive drawing masks for the pointer.

PointerSize *(BOOL)* — input
Pointer-size indicator:

TRUE The bit map should be stretched (if necessary) to the system pointer dimensions
FALSE The bit map should be stretched (if necessary) to the system icon dimensions.

xHotspot *(SHORT)* — input
x offset of hotspot within pointer from its lower left corner (in pels).

yHotspot *(SHORT)* — input
y offset of hotspot within pointer from its lower left corner (in pels).

hptr *(HPOINTER)* — return
Pointer handle:

NULL Error
Other Handle of the newly created pointer.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_HBITMAP_BUSY

Remarks

A pointer can be created either as a true pointer (at pointer size), or as an icon pointer (at icon size). The latter is useful when using Icons as direct-manipulation objects that the user can "pick up" and move about the screen as a means of performing some operation. (See also WinCreatePointerIndirect.)

This call requires the existence of a message queue.

WinCreatePointerIndirect – Create Pointer Indirect

This call creates a colored pointer or icon from a bit map.

```
WinCreatePointerIndirect (DeskTop, PointerInfo, hptr)
```

Parameters

DeskTop *(HWND)* – input
Desktop-window handle or HWND_DESKTOP.

PointerInfo *(POINTERINFO)* – input
Pointer information structure.

The fields in this structure must be set before the call is made:

- **flag** is set to TRUE if a pointer is being created, or to FALSE if an icon is being created,
- **xhotspot** and **yhotspot** are set to the relative position in the icon or pointer that is associated with the mouse position,
- **hbitmap** is a bit-map that specifies the AND and XOR masks, as used for black and white pointers and icons, and
- **color** is a color bit-map that describes the color content of the pointer or icon.

It is an error if **hbitmap** is NULL. Also, the width of **hbitmap** must be the same as that of **color**, and the height of **hbitmap** must be double that of **color** (to allow for both the AND and the XOR mask).

hptr *(HPOINTER)* – return
Pointer handle:

NULL Error
Other Handle of the newly created pointer or icon.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_HBITMAP_BUSY

Remarks

A pointer can be created either as a true pointer (at pointer size), or as an icon pointer (at icon size). The latter is useful when using icons as direct-manipulation objects that the user can "pick up" and move about the screen as a means of performing some operation. (See also WinCreatePointer.)

This call requires the existence of a message queue.

WinCreateStdWindow — Create Standard Window

This call creates a standard window.

WinCreateStdWindow (Parent, Style, CreateFlags, ClassClient, Title, StyleClient, Resource, Id, Client, *Frame*)

Parameters

Parent *(HWND)* — input
Parent-window handle.

If this parameter is a window handle returned from the WinQueryDesktopWindow function, or is HWND_DESKTOP, a main window is created.

If **Parent** is a window handle returned from WinQueryObjectWindow, or is HWND_OBJECT, an object window is created.

Style *(BIT32)* — input
Frame-window style.

This is a combination of any of the WS_* styles (see "Window Styles" on page 12-2) and the FS_* (see Frame Window Styles on page 15-3) frame styles.

The interpretation of the parameters is affected by the use of these styles.

CreateFlags *(BIT32)* — input
Frame-creation flags.

This contains a combination of any of the FCF_* flags.

The interpretation of the parameters is affected by the use of these flags; see page 15-2.

ClassClient *(STRL)* — input
Client-window class name.

If **ClassClient** is not a zero-length string, a client window of style **StyleClient** and class **ClassClient** is created. **ClassClient** is either an application specified name as defined by WinRegisterClass or the name of a preregistered WC_* class; see page 11-1. Preregistered class names are of the form '#nnnnn', where nnnnn is 1 through 5 digits corresponding to the value of the WC_* class name constant.

If **ClassClient** is NULL, no client area is created either.

This parameter can also be specified directly as a WC_* constant.

Title *(STRL)* — input
Title-bar text.

This is ignored if FCF_TITLEBAR (or FCF_STANDARD) is not specified in **CreateFlags**.

StyleClient *(BIT32)* — input
Client-window style.

This is ignored if **ClassClient** is a zero-length string.

Resource *(RESID)* — input
Resource identifier.

This is ignored unless FCF_MENU, FCF_STANDARD, FCF_ACCELTABLE, or FCF_ICON is specified.

NULL Resource definitions are contained in the application .EXE file.
Other The module handle returned by the DosLoadModule or DosGetModHandle call of the Dynamic Link Library (DLL) containing the resource definitions.

WinCreateStdWindow — Create Standard Window

SAA

Id *(IDENTITY)* — input
Frame-window identifier.

The identifier within the resource definition of the required resource.

It is the responsibility of the application to ensure that all of the resources related to one frame window have the same **Id** value.

Client *(HWND)* — output
Client-window handle.

This is returned if a client window is created.

Frame *(HWND)* — return
Frame-window handle.

This is NULL if no window is created.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG
PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND

Remarks

The window is created with zero width and depth and positioned at the bottom left of the **Parent**, unless FCF_SHELLPOSITION is specified in which case the size and position are set by the shell. The window can be positioned and sized by use of WinSetWindowPos.

If WS_VISIBLE is set, the frame window is created visible. It is recommended that standard windows that are not main windows are created with WS_VISIBLE not set.

Frame is the window handle of the frame window, that is, the window of class WC_FRAME, and has a parent of **Parent**.

The frame window is created with identity **Id**, all the component windows, known as the frame controls, have the standard window identifiers FID_*; see page 15-1. The identifier FID_CLIENT is used for the client area of the window.

It may be necessary to change the **Id** of the frame window after it has been created, so that another frame window can be created with the same resource tables, and still maintain distinct window identities. This can be achieved with the WinSetWindowUShort call.

Some combinations of frame control flags are valid, but leave visual holes in the frame window. Specifically, if the **CreateFlags** parameter specifies any of FCF_SYSMENU, FCF_MINBUTTON, FCF_MAXBUTTON or FCF_MINMAX, but not FCF_TITLEBAR, the area of the top title line between the optional system menu and the minimize/maximize icons is not drawn by the default frame window procedure.

None of the following can be used with WinCreateStdWindow:

- WS_CLIPCHILDREN for the frame style
- WS_CLIPSIBLINGS for the style of the client window or any of the frame control windows
- CS_CLIPSIBLINGS for the class style of the window.

If any of the above are specified, the window is not redrawn correctly. Any style can be used for the children of the client. If it really is required that a client or a frame control is CLIPSIBLINGS, the application must ensure that it is in front of the client and all the other frame controls, for it to be drawn.

This call requires the existence of a message queue.

This call adds an entry to the task list.

WinCreateSwitchEntry (hab, SwitchData, *Switch*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

SwitchData *(SWCNTRL)* – input
Switch data.

Contains information about the newly-created task-list entry.

If the **swtitle** field of the *SWCNTRL* structure is NULL, the system uses the name under which the application is started. This only applies for OS/2 Version 1.2 programs, and only for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

Leading and trailing blanks are removed from the title, and if necessary it is truncated to 60 characters.

If the **hprog** field of the *SWCNTRL* structure is NULL, the value used by the system when the program was loaded (if it has been loaded) is substituted.

If the **process** field of the *SWCNTRL* structure is NULL, the current process ID is used.

Switch *(HSWITCH)* – return
Handle to the newly-created task-list entry.

There is a system limit to the number of task-list entries. However, this is a large number (several hundred) and is unlikely to be reached in practice since other system limits, such as memory size impinge first.

NULL Error occurred
Other Handle to the newly-created task-list entry.

Possible returns from WinGetLastError:

PMERR_NO_SPACE
PMERR_INVALID_WINDOW
PMERR_INVALID_SESSION_ID

Remarks

Both this call and the WinRemoveSwitchEntry call are not required if the main window is created with the frame creation flags FCF_TASKLIST or FCF_STANDARD, as these styles automatically update the task list when the main window is created, destroyed, or its title changes. (See also WinAddSwitchEntry.)

This call requires the existence of a message queue.

WinCreateWindow — Create Window

SAA

This call creates a new window of class **ClassName** and returns **hwnd**.

WinCreateWindow (Parent, ClassName, Name, Style, xcoord, ycoord, Width, Height, Owner, Behind, Id, CtlData, PresParams, *hwnd***)**

Parameters

Parent *(HWND)* — input
Parent-window handle.

If **Parent** is a desktop window handle, or is HWND_DESKTOP, a main window is created.

If **Parent** is HWND_OBJECT, or is a window handle returned by WinQueryObjectWindow, an object window is created.

ClassName *(STRL)* — input
Registered-class name.

ClassName is either an application specified name as defined by WinRegisterClass or the name of a preregistered WC_* class; see page 11-1. Preregistered class names are of the form '#nnnnn', where nnnnn is 1 through 5 digits corresponding to the value of the WC_* class-name constant.

This parameter can also be specified directly as a WC_* constant.

Name *(STRL)* — input
Window text.

The actual structure of the data is class-specific. It is usually a null-terminated string that is often displayed in the window.

Style *(BIT32)* — input
Window style.

xcoord *(SHORT)* — input
x coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

ycoord *(SHORT)* — input
y coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

Width *(SHORT)* — input
Width of window, in window coordinates.

Height *(SHORT)* — input
Height of window, in window coordinates.

Owner *(HWND)* — input
Owner-window handle.

Windows that send messages send them to their owner, as defined by this parameter. When an owner window is destroyed, all windows owned by it are also destroyed. The owner window must belong to the current thread.

Behind *(HWND)* — input
Sibling-window handle.

This is the sibling window behind which **hwnd** is placed. If this parameter is HWND_TOP or HWND_BOTTOM, **hwnd** is placed on top of all, or behind all of its siblings. This parameter must be HWND_TOP, HWND_BOTTOM, or a child of **Parent**.

Id *(IDENTITY)* — input
Window identifier.

A value given by the application that enables specific children of a window to be identified. For example, the controls of a dialog have unique identifiers so that an owner can distinguish which control has notified it. Window identifiers are also used for frame windows.

CtlData *(CTLDATA)* — input
Control data.

This is class-specific data passed to the window procedure by the WM_CREATE message.

PresParams *(PRESDATA)* — input
Presentation parameters.

This is class-specific presentation data passed to the window procedure by the WM_CREATE message.

hwnd *(HWND)* — return
Window handle:

NULL Error occurred
Other Window handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG
PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND

Remarks

The appearance and behavior of a window are determined by its style, which is the (logical-OR) combination of the style established by **ClassName** and **Style**. Any of the standard styles WS_* (see "Window Styles" on page 12-2) can be used in addition to any class-specific styles that may be defined.

A window is usually created enabled and invisible. For more information on the initial state of a created window, see the list of the standard window styles.

Messages may be received from other processes or threads when this function is called.

This function sends the WM_CREATE message to the window procedure of the window being created.

This function sends the WM_ADJUSTWINDOWPOS message after the WM_CREATE message, and before the window is displayed (if applicable). The values passed are those given to the WinCreateWindow function. If the window has style WS_VISIBLE, the window is created visible.

The WM_SIZE message is not sent by the WinCreateWindow function while the window is being created. Any required size processing can be performed during the processing of the WM_CREATE message.

Because windows are often created with zero height or width and sized later, it is good practice not to perform any size-related processing if the size of the window is zero.

If the WinCreateWindow function is called for a window of class WC_FRAME, the controls specified by **createflags** are created but not formatted. The frame is formatted when a WM_FORMATFRAME message is received but this is not sent during window creation. To cause the frame to format, either a WM_FORMATFRAME message should be sent, or the window position adjusted using the WinSetWindowPos function call which sends a WM_SIZE message if the position or size is changed.

The only limitations to the size and position specified for a window are the number range allowed for the size and position parameters; that is, an application can create windows that are larger than the screen or that are positioned partially or totally off the screen. However, the user interface for manipulation of window sizes and positions is affected if part or all of the window is off the screen.

It is recommended that part of the title bar is left on the screen, if the window has one, to enable the user to move the window around.

When a WC_MENU window is created with this call, **CtlData** is assumed to be a menu template, which is used to create the menu. If **CtlData** is NULL, an empty menu is created.

This call requires the existence of a message queue.

WinDdeInitiate – Dynamic Data Exchange Initiate

This call is issued by a client application to one or more other applications, to request initiation of a dynamic data exchange conversation.

WinDdeInitiate (Client, AppName, TopicName, *Success***)**

Parameters

Client *(HWND)* – input
Client's window handle.

This window will typically not be visible.

AppName *(STRL)* – input
Application name.

This is the name of the desired server application. If it is a zero-length string, any application can respond.

Application names may not contain slashes or backslashes.

TopicName *(STRL)* – input
Topic name.

This is the name of the desired topic. If it is a zero-length string, each responding application will respond once for each topic which it can support.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion. The WM_DDE_INITIATE message is successfully sent to all appropriate windows.
FALSE Error occurred.

Remarks

This call sends a WM_DDE_INITIATE message to all top level frame windows. These are windows registered with CS_FRAME, whose parent is the desktop window. No message is sent to object windows.

The WinDdeInitiate call does not return to the client application until all receiving applications have, in sequence, processed the WM_DDE_INITIATE message, and the client application has received all the corresponding WM_DDE_ACK messages. (See WinDdeRespond.)

This call requires the existence of a message queue.

WinDdePostMsg — Dynamic Data Exchange Post Message

This call is issued by an application to post a message to another application with which it is carrying out a dynamic data exchange conversation.

WinDdePostMsg (To, From, Msgid, Data, Retry, *Success*)

Parameters

To *(HWND)* — input
Window handle of target.

From *(HWND)* — input
Window handle of originator.

Msgid *(USHORT)* — input
Message identifier.

Identifies the message to be posted.

The following messages are valid:

WM_DDE_ACK
WM_DDE_ADVISE
WM_DDE_DATA
WM_DDE_EXECUTE
WM_DDE_POKE
WM_DDE_REQUEST
WM_DDE_TERMINATE
WM_DDE_UNADVISE

Data *(DDESTRUCT)* — input
DDE structure.

Retry *(BOOL)* — input
Retry indicator.

This controls what happens if the message cannot be posted because the destination queue is full.

Programming Note: If the message posting fails for any other reason (for example, an invalid window handle is specified), this call returns FALSE whatever the value of **Retry**.

TRUE The message posting is retried at 1-second intervals, until the message is posted successfully.

In this case, WinDdePostMsg dispatches any messages in the queue, by calling WinPeekMsg and WinDispatchMsg in a loop. Messages sent by other applications can also be received. This means that the application can continue to receive DDE messages (or other kinds of messages), while attempting to post DDE messages.

FALSE The call returns FALSE without retrying.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call requires the existence of a message queue.

WinDdeRespond – Dynamic Data Exchange Respond

This call is issued by a server application to indicate that it can support a dynamic data exchange conversation on a particular topic.

WinDdeRespond (Client, Server, AppName, TopicName, *Reply*)

Parameters

Client *(HWND)* – input
Client's window handle.

Server *(HWND)* – input
Server's window handle.

If a server application is responding for more than one topic, it must use a different window for each topic.

AppName *(STRL)* – input
Application name.

This is the name of the responding server application. It must not be a zero-length string.

Application names may not contain slashes or backslashes.

TopicName *(STRL)* – input
Topic name.

This is the name of the topic which the server is willing to support. It must not be a zero-length string.

Reply *(MRESULT)* – return
Message return data.

Remarks

This call is issued by a server application after receiving a WM_DDE_INITIATE message that identifies this server application (or indicates that any application can respond), and also either identifies a particular topic which the server can support, or asks for all supported topics (see WinDdeInitiate). WinDdeRespond sends a WM_DDE_INITIATEACK message back to the client, that is the sender of the WM_DDE_INITIATE message.

If the server application can respond, it issues a WinDdeRespond call once if a specific topic was requested, or once for each topic which it can support if all supported topics were requested.

A DDE conversation is initiated each time this call is successfully issued. The client is expected to terminate all unwanted conversations. Once a conversation is initiated, it is controlled by the client issuing WinDdePostMsg calls.

This call requires the existence of a message queue.

WinDefAVioWindowProc — Default AVio Window Procedure

This call invokes the default AVIO window procedure.

WinDefAVioWindowProc (hwnd, Msgid, Param1, Param2, *reply*)

Parameters

hwnd *(HWND)* — input
Window handle.

Msgid *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input
Parameter 1.

Param2 *(MPARAM)* — input
Parameter 2.

reply *(MRESULT)* — return
Message return data.

Remarks

Applications using AVIO must pass all WM_SIZE messages for the window with which the AVIO presentation space is associated to this routine, using the same parameters as are received in the WM_SIZE message. This routine maintains the window size data in the presentation space, and must be called before the application accesses the window.

This call is not a replacement for WinDefWindowProc, which must also be called as usual, to process any messages that have not been handled by the application's window procedure.

This call requires the existence of a message queue.

WinDefDlgProc — Default Dialog Procedure

SAA

This call invokes the default dialog procedure with **Dlg**, **Msgid**, **Param1**, and **Param2**.

WinDefDlgProc (Dlg, Msgid, Param1, Param2, *Reply***)**

Parameters

Dlg *(HWND)* — input
Dialog-window handle.

Msgid *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input
Parameter 1.

Param2 *(MPARAM)* — input
Parameter 2.

Reply *(MRESULT)* — return
Message-return data.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The action taken by the default dialog procedure is such that the values passed in **Param1** and **Param2**, and the values returned in **Reply** are defined for each **Msgid**.

The default dialog procedure provides default processing for any dialog window messages that an application chooses not to process. It can be used to ensure that every message is processed and is called with the same parameters that were received by the dialog procedure.

The action of the WinDefDlgProc call on receiving messages is precisely the same as for the frame window procedure. If an application processes a message instead of sending it to the WinDefDlgProc call, it may be required to perform some or all of the frame window procedure actions for itself.

This call requires the existence of a message queue.

WinDefWindowProc — Default Window Procedure

This call invokes the default window procedure.

WinDefWindowProc (hwnd, Msgid, Param1, Param2, *Reply***)**

Parameters

hwnd *(HWND)* — input
Window handle.

Msgid *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input
Parameter 1.

Param2 *(MPARAM)* — input
Parameter 2.

Reply *(MRESULT)* — return
Message-return data.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The default window provides default processing for any window messages that an application chooses not to process. It can be used to ensure that every message is processed. This call should be made with the same parameters as those received by the window procedure.

The action taken by the default window procedure, the values passed in **Param1**, **Param2** and the values returned in **Reply** are defined for each **Msgid**. (See Chapter 11, "Introduction to Message Processing.")

This call requires the existence of a message queue.

WinDeleteAtom — Delete Atom

This call deletes an atom from an atom table.

```
WinDeleteAtom (AtomTbl, atom, ReturnCode)
```

Parameters

AtomTbl *(HATOMTBL)* — input
Atom-table handle.

This is the handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable call.

atom *(ATOM)* — input
Atom identifying the atom to be deleted.

ReturnCode *(ATOM)* — return
Return code:

0 Call successful
Other The call fails and the atom has not been deleted, in which case this is equal to the **atom** parameter.

Possible returns from WinGetLastError:

PMERR_INVALID_HATOMTBL
PMERR_INVALID_ATOM

Remarks

If the passed atom is an integer atom, 0 is returned. If it is not an integer atom and it is a valid atom for the given atom table, that is, it has an atom name and use count, its use count is decremented by one and 0 is returned. If the use count has been decremented to zero, the atom name and use count are removed from the atom table.

WinDeleteLibrary — Delete Library

This call deletes the library **Libhandle**.

WinDeleteLibrary (hab, Libhandle, *Deleted***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Libhandle *(HLIB)* — input
Library handle to be deleted.

Deleted *(BOOL)* — return
Library-deleted indicator.

TRUE Library successfully deleted
FALSE Library not successfully deleted.

Remarks

This call requires the existence of a message queue.

WinDeleteProcedure — Delete Procedure

This call deletes the window or dialog procedure.

WinDeleteProcedure (hab, wndproc, *success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

wndproc *(WNDPROC)* — input
Window procedure identifier to be deleted.

success *(BOOL)* — return
Procedure-deleted indicator:

TRUE Procedure successfully deleted
FALSE Procedure not successfully deleted.

Remarks

This call requires the existence of a message queue.

WinDestroyAccelTable — Destroy Accelerator Table

This call destroys an accelerator table.

WinDestroyAccelTable (Accel, *Success*)

Parameters

Accel *(HACCEL)* — input
Accelerator-table handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HACCEL

Remarks

This call requires the existence of a message queue.

WinDestroyAtomTable — Destroy Atom Table

This call destroys an atom table.

```
WinDestroyAtomTable (AtomTbl, ReturnCode)
```

Parameters

AtomTbl *(HATOMTBL)* — input
Atom-table handle.

This is the handle returned from a previous call to the WinCreateAtomTable call. If NULL then this call does nothing.

ReturnCode *(HATOMTBL)* — return
Return code:

0 Function successful.
Other The call fails and the atom table has not been destroyed, in which case this is equal to the **AtomTbl** parameter.

Possible returns from WinGetLastError:

PMERR_INVALID_HATOMTBL

Remarks

This call makes no attempt to ensure that the handle to the atom table is not reused by a later call to the WinCreateAtomTable call.

The system atom table (see the WinQuerySystemAtomTable call) cannot be destroyed.

WinDestroyCursor — Destroy Cursor

This call destroys the current cursor, if it belongs to the specified window.

WinDestroyCursor (hwnd, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle to which the cursor belongs.

This can be the desktop-window handle or HWND_DESKTOP.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call has no effect if the current cursor does not belong to the specified window.

This call requires the existence of a message queue.

WinDestroyDataStructure — Destroy Data Structure

SAA

This call destroys a data structure that was created using the WinCreateDataStructure call, or whose handle was returned to the application by PM (for example, by the WinQueryDataStructure call, the WinGetMsg call, or contained within a structure which is an output or input-output function parameter).

WinDestroyDataStructure (hab, Handle, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Handle *(HSTRUCT)* — input/output
Data structure handle.

This is the handle of the structure that is to be destroyed.

If the handle is an alias, the alias becomes undefined, but the data structure and the original handle continue to exist.

If the handle is the original handle returned when the structure was created, the structure is destroyed and any aliases for it become invalid; they must be destroyed using the WinDestroyDataStructure call.

PM always returns a value of NULL.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB
PMERR_INV_HSTRUCT

Remarks

It is the application's responsibility to destroy all data structures created by it, and all data-structure handles returned to it by PM. Failure to do this may result in storage errors.

For the special case of message parameters returned by the WinGetMsg and WinPeekMsg calls, the WinFreeMsg call should be used, as this relieves the application of the need to know whether the message parameter was a data-structure handle or a simple data type.

Destroying a handle which is an alias leaves the original structure and handle intact; destroying the original handle will cause the alias to become invalid. Program errors may result if it is used on any call other than WinDestroyDataStructure.

Programming Note: This function is required only in COBOL and FORTRAN.

This call requires the existence of a message queue.

WinDestroyHeap – Destroy Heap

This call destroys a heap, created by the WinCreateHeap call.

WinDestroyHeap (Heap, *HeapRet*)

Parameters

Heap *(HHEAP)* – input
Heap handle.

Returned by previous call to the WinCreateHeap call. If NULL, this call does nothing.

HeapRet *(HHEAP)* – return
Returned heap handle:

NULL Successful completion
Other Returned heap handle.

Remarks

If the DosAllocSeg call is called by the WinCreateHeap call to allocate space for the heap, then this call calls the DosFreeSeg call to free the allocated segment. Otherwise, this call only frees the passed heap handle.

Allocated memory objects within the heap are not significant to this call.

This call makes no attempt to ensure that the handle to the heap being destroyed as specified by the **Heap** parameter is not recreated by a later use of the WinCreateHeap call.

WinDestroyHelpInstance — Destroy Help Instance

This call destroys the specified instance of the help manager.

```
WinDestroyHelpInstance (HelpInstance, Success)
```

Parameters

HelpInstance *(HWND)* — input
Handle of the instance of the help manager to be destroyed.

This is the handle returned by the WinCreateHelpInstance call.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call requires the existence of a message queue.

WinDestroyMsgQueue — Destroy Message Queue

This call destroys the message queue.

WinDestroyMsgQueue (hmq, *Destroyed*)

Parameters

hmq *(HMQ)* — input
Message-queue handle.

Destroyed *(BOOL)* — return
Queue-destroyed indicator:

TRUE Queue destroyed
FALSE Queue not destroyed.

Possible returns from WinGetLastError:

PMERR_INVALID_HMQ

Remarks

This call must be made before terminating a thread or an application.

This call requires the existence of a message queue.

WinDestroyPointer — Destroy Pointer

SAA

This call destroys a pointer or icon.

```
WinDestroyPointer (Pointer, Success)
```

Parameters

Pointer *(HPOINTER)* — input
Handle of pointer to be destroyed.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HPTR

Remarks

A pointer can only be destroyed by the thread that created it.

The system pointers and icons must not be destroyed.

This call requires the existence of a message queue.

WinDestroyWindow — Destroy Window

This call destroys a window and its child windows.

WinDestroyWindow (hwnd, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle.

Success *(BOOL)* — return
Window-destroyed indicator:

TRUE Window destroyed
FALSE Window not destroyed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The window to be destroyed must have been created by the thread that is issuing this call. Before **hwnd** is itself destroyed, all windows owned by **hwnd** are also destroyed.

If **hwnd** cannot be destroyed, for example because **hwnd** is an invalid window handle or is not associated with the current thread, **Success** returns FALSE.

Programming Note: If **hwnd** is locked, this call does not return until the window is unlocked. (See the WinLockWindow call.)

Messages may be received from other processes or threads during the processing of this call.

If a Presentation Space is associated with the window, it is disassociated from it by this function. If the presentation space was obtained by use of the GpiCreatePS function, then it is disassociated from the window, but not destroyed, that is this function performs the GpiAssociate function to disassociate the presentation space but does not perform the GpiDestroyPS function. If the presentation space was obtained by use of the WinGetPS function, it is released by this function; that is, this function performs the WinReleasePS function.

Messages sent by this call are:

WM_DESTROY	Always sent to the window being destroyed after the window has been hidden on the device, but before its children have been destroyed. The message is sent first to the window being destroyed, then to the children as they are destroyed. Therefore, during processing the WM_DESTROY it can be assumed that all the children still exist.
WM_ACTIVATE	Sent with **active** set to FALSE if the window being destroyed is the active window.
WM_OTHERWINDOWDESTROYED	Sent to all main windows if **hwnd** or any of its descendants has been registered with WinRegisterWindowDestroy.
WM_RENDERALLFMTS	Sent if the clipboard owner is being destroyed, and there are unrendered formats in the clipboard.

WinDestroyWindow — Destroy Window

SAA

If the window being destroyed is the active window, both the active window and the input focus window are transferred to another window when the window is destroyed. The window that becomes the active window is the next window, as defined for the 'Alt+Esc' function. This usually corresponds to the next application in the sequence. The input focus transfers to whichever window the new active window decides should have it.

This call requires the existence of a message queue.

WinDismissDlg — Dismiss Dialog

This call hides the modeless dialog window, or destroys the modal dialog window, and causes the WinProcessDlg or WinDlgBox calls to return.

WinDismissDlg (Dlg, Result, *Success*)

Parameters

Dlg *(HWND)* — input
Dialog-window handle.

Result *(USHORT)* — input
Reply value.

Returned to the caller of the WinProcessDlg or WinDlgBox calls.

Success *(BOOL)* — return
Dialog-dismissed indicator:

TRUE Dialog successfully dismissed
FALSE Dialog not successfully dismissed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is required to complete the processing of a modal dialog window and is called from its dialog procedure. The call is made implicitly if the dialog procedure passes a WM_COMMAND message to WinDefDlgProc or if a WM_QUIT message is encountered during a WinProcessDlg or WinGetDlgMsg call.

This call hides the dialog window, end re-enables any windows that were disabled by a WinProcessDlg or WinGetDlgMsg call.

This call does not destroy the dialog window. A WinDestroyWindow call must be issued to destroy the dialog window when it is no longer needed. However, the WinDlgBox call destroys the dialog window that it creates, when the dialog window is dismissed by the use of this call.

This call can be issued during the processing of the WM_INITDLG message.

Programming Note: This call can be made from a modeless dialog window, although this is not necessary as there is no internal message processing loop. If it is called, the dialog window is hidden. It is the responsibility of the application to destroy the dialog window, if required.

This call requires the existence of a message queue.

WinDispatchMsg — Dispatch Message

SAA

This call invokes a window procedure.

WinDispatchMsg (hab, Msg, *Reply*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Msg *(QMSG)* — input
Message structure.

Reply *(MRESULT)* — return
Message-return data.

Remarks

This call is equivalent to using the WinSendMsg call with the parameters corresponding to those in **Msg**.

The time and pointer position information within **Msg** can be obtained by the window procedure with the WinQueryMsgTime and WinQueryMsgPos calls.

Reply is the value returned by the invoked window procedure. For standard window classes, the values of **Reply** are documented with the message definitions; see Chapter 11, "Introduction to Message Processing."

This call requires the existence of a message queue.

WinDlgBox — Load and Process Modal Dialog

This call loads and processes a modal dialog window and returns the result value established by the WinDismissDlg call.

WinDlgBox (Parent, Owner, DlgProc, Resource, Dlgid, CreateParams, *Result*)

Parameters

Parent *(HWND)* — input
Parent-window handle of the created dialog window:

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

Owner *(HWND)* — input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg call.

DlgProc *(WNDPROC)* — input
Dialog procedure for the created dialog window.

Resource *(RESID)* — input
Resource identity containing the dialog template.

NULL	Use the application's .EXE file.
Other	Module handle returned from the DosLoadModule or DosGetModHandle call.

Dlgid *(USHORT)* — input
Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window.

CreateParams *(CREATEPARAMS)* — input
Application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

Result *(USHORT)* — return
Reply value.

Value established by the WinDismissDlg call or DID_ERROR if an error occurs.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND
PMERR_RESOURCE_NOT_FOUND

Remarks

The use of parameters to this call are the same as those of the WinLoadDlg call.

This call should not be used while pointing device capture is set. (See WinSetCapture.)

This call does not return until WinDismissDlg is called.

WinDlgBox — Load and Process Modal Dialog

SAA

This call is equivalent to:

```
WinLoadDlg (., ., ., ., ., ., dlg);
WinProcessDlg (dlg, result);
WinDestroyWindow (dlg, success);
return  (result);
```

and the remarks documented under these calls also apply.

Programming Note: This can be considered to be a customizable "read from screen" call. The caller supplies a data buffer (the **CreateParams** parameter), filled with initial values. It receives a return code which indicates whether the data in the buffer has been updated and validated, or whether the end user canceled the dialog.

The end user interface is encapsulated within the dialog window. The dialog template provides a view of the current state of the data buffer, the dialog procedure defines how the user can change the data.

The caller need know nothing about the details of the end user interface. It makes a single "read from screen" call and continues with its work.

This call requires the existence of a message queue.

WinDrawBitmap — Draw Bit Map

This call draws a bit map using the current image colors and mixes.

WinDrawBitmap (hps, hbm, Src, Dest, ForeColor, BackColor, Rgf, *Success***)**

Parameters

hps *(HPS)* — input
Handle of presentation space in which the bit map is drawn.

hbm *(HBITMAP)* — input
Bit-map handle.

Src *(RECT)* — input
Subrectangle of bit map to be drawn:

Programming Note: The data type *WRECT* may also be used, if supported by the language.

NULL The whole of the bit map is drawn
Other The whole of the bit map is not drawn.

Dest *(POINT)* — input
Bit-map destination.

The bottom left hand corner of the bit-map destination is specified in device coordinates.

ForeColor *(LONG)* — input
Foreground color.

This is used if **hbm** refers to a monochrome bit map. In this instance, bit-map bits that are set to 1 are drawn using **ForeColor**. Ignored if DBM_IMAGEATTRS is specified.

BackColor *(LONG)* — input
Background color.

This is used if **hbm** refers to a monochrome bit map. In this instance, bit-map bits that are set to zero are drawn using **BackColor**. Ignored if DBM_IMAGEATTRS is specified.

Rgf *(BIT16)* — input
Flags that determine how the bit map is drawn:

DBM_NORMAL	Draw the bit map normally using ROP_SRCCOPY, as defined in GpiBitBlt.
DBM_INVERT	Draw the bit map inverted using ROP_NOTSRCCOPY, as defined in GpiBitBlt.
DBM_STRETCH	**Dest** points to a RECT data structure, representing a rectangle in the destination presentation space, to which the bit map should be stretched or compressed. If compression is required, some rows and columns of the bit map are eliminated.
DBM_IMAGEATTRS	If this is specified, color conversion of monochrome bit maps is done by using the image attributes.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_FLAG
PMERR_HBITMAP_BUSY

WinDrawBitmap — Draw Bit Map

Remarks

This call should only be used in draw mode (DM_DRAW) to a screen device context (see GpiSetDrawingMode). The presentation space handle can be to either a micro-presentation space or a normal presentation space (see GpiCreatePS).

If **hbm** refers to a color bit map, no color conversion is performed.

The current position in the presentation space is not changed by this call.

WinDrawBorder — Draw Border

This call draws the borders and interior of a rectangle.

WinDrawBorder (hps, Rectangle, VertSideWidth, HorizSideWidth, BorderColor, InteriorColor, Cmd, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Rectangle *(RECT)* — input
Bounding rectangle for the border.

The rectangle is in device coordinates.

The border is drawn within the rectangle. Along the bottom and left edges of the rectangle, the edges of the border coincide with the rectangle edges. Along the top and right edges of the rectangle, the border is drawn one device unit inside the rectangle edges.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

VertSideWidth *(SHORT)* — input
Width of border rectangle vertical sides.

This is the width of the left and right sides in device coordinates.

HorizSideWidth *(SHORT)* — input
Width of border rectangle horizontal sides.

This is the width of the top and bottom sides in device coordinates.

BorderColor *(LONG)* — input
Color of edge of border.

Not used if DB_AREAATTRS is specified.

InteriorColor *(LONG)* — input
Color of interior of border.

Not used if DB_AREAATTRS is specified.

Cmd *(BIT16)* — input
Flags controlling the way in which the border is drawn.

Some of the DB_ flags are mutually exclusive. Only one of these is significant:

- DB_PATCOPY (default)
- DB_PATINVERT
- DB_DESTINVERT
- DB_AREAMIXMODE.

DB_ROP
A group of flags that specify the mix to be used, for both the border and the interior.

DB_PATCOPY
Use the ROP_PATCOPY raster operation (see GpiBitBlt). This is a copy of the pattern to the destination.

DB_PATINVERT
Use the ROP_PATINVERT raster operation (see GpiBitBlt). This is an exclusive-OR of the pattern with the destination.

DB_DESTINVERT
Use the ROP_DESTINVERT raster operation (see GpiBitBlt). This inverts the destination.

DB_AREAMIXMODE
Map the current area foreground mix attribute into a Bitblt raster operation (see GpiBitBlt). The area background mix mode is ignored.

DB_INTERIOR
The area contained within the given rectangle, and not included within the borders (as given by **VertSideWidth** and **HorizSideWidth**), is drawn.

DB_AREAATTRS

- If this is specified:

  For any border, the pattern used is the pattern as currently defined in the area attribute.

  For any interior, the pattern used is the same as if a GpiSetAttrs call for the area attributes is made with the area attribute's background color being passed for the foreground color, and the area attribute's foreground color being passed as the background color.

- If this is not specified (default):

  For any border, the pattern used is the same as if a GpiSetAttrs call for the area attributes is made with a foreground color of **BorderColor**, and a background color of **InteriorColor**.

  For any interior, the pattern used is the same as if a GpiSetAttrs call for the area attributes is made with a foreground color of **InteriorColor**, and a background color of **BorderColor**.

DB_STANDARD
VertSideWidth and **HorizSideWidth** are multiplied by the system SV_CXBORDER and SV_CYBORDER constants to produce the widths of the vertical and horizontal sides of the border.

DB_DLGBORDER
A standard dialog border is drawn, in the active title bar color if DB_PATCOPY is specified, or the inactive title bar color if DB_PATINVERT is specified. Other DB_ROP options, and DB_AREAATTRS, are ignored.

DB_ROP and DB_AREAATTRS are also ignored for the interior. The interior is drawn in the color specified by **InteriorColor**.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_FLAG
PMERR_INV_DRAW_BORDER_OPTION

Remarks

A border is a rectangular frame, normally used around the edge of a window.

This call should only be used in draw mode (DM_DRAW), to a screen device context; **hps** can be either a micro-presentation space or a normal presentation space (see GpiCreatePS). DB_DESTINVERT inverts the destination.

If DB_AREAMIXMODE is given, the foreground mix mode from the area attribute is mapped into an equivalent ROP_ value (see GpiBitBlt). The area background mix mode is ignored.

Either or both **VertSideWidth** or **HorizSideWidth** may be given as zero. If both are given as zero, the interior is still drawn. If either the x borders overlap or the y borders overlap, the border is drawn as a single rectangle with no interior.

WinDrawPointer — Draw Pointer

This call draws a pointer.

WinDrawPointer (hps, x, y, Pointer, Halftone, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle into which the pointer is drawn.

This can be either a micro presentation space or a normal presentation space (see GpiCreatePS).

x *(SHORT)* — input
x coordinate at which to draw the pointer, in device coordinates.

y *(SHORT)* — input
y coordinate at which to draw the pointer, in device coordinates.

Pointer *(HPOINTER)* — input
Pointer handle.

This is equivalent to a bit-map handle and is returned from calls such as the GpiLoadBitmap call.

Halftone *(USHORT)* — input
Shading control:

DP_NORMAL	Draw the pointer as it normally appears.
DP_HALFTONED	Draw the pointer with a halftone pattern where black normally appears.
DP_INVERTED	Draw the pointer inverted, black for white and white for black.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Function failed.

Possible returns from WinGetLastError:

PMERR_INVALID_HPTR
PMERR_INVALID_FLAG

Remarks

This call should only be used in draw mode (DM_DRAW) to a screen device context.

WinDrawText — Draw Text

SAA

This call draws a single line of formatted text into a specified rectangle.

WinDrawText (hps, Count, Text, Rectangle, ForeColor, BackColor, Cmd, *Chars*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Count *(SHORT)* — input
Count of the number of characters in the string:

−1 The string is null terminated and its length is to be calculated by this function.
Other Count of the number of characters in the string.

Text *(STRCOND)* — input
Character string to be drawn.

A carriage-return or line-feed character terminates the line, even if the line is less than **Count**.

Rectangle *(RECT)* — input/output
Text rectangle.

Rectangle within which the text is to be formatted, in world coordinates. Points on the boundary of this rectangle are deemed to be inside the rectangle.

The return value is only of interest in the instance where DT_QUERYEXTENT is set.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

ForeColor *(LONG)* — input
Foreground color.

Ignored if DT_TEXTATTRS is specified.

BackColor *(LONG)* — input
Background color.

The background is drawn with the current background mix. The default is BM_LEAVEALONE, that is, **BackColor** is ignored unless GpiSetBackMix is called.

The background rectangle is the rectangle that bounds the text; it is not the input parameter rectangle.

This parameter is ignored if DT_TEXTATTRS is specified.

Cmd *(BIT16)* — input
An array of flags that determines how the text is drawn.

Some of the DT_ flags are mutually exclusive. Only **one** from each of these groups is significant:

- DT_LEFT (default), DT_CENTER, DT_RIGHT
- DT_TOP (default), DT_VCENTER, DT_BOTTOM.

When mutually-exclusive flags are used together, the function gives indeterminate results.

DT_WORDBREAK can only be specified with DT_TOP and DT_LEFT.

If DT_HALFTONE, DT_ERASERECT, or DT_MNEMONIC is used, the presentation space must be in PU_PELS units.

DT_LEFT	Left-justify the text.
DT_CENTER	Center the text.
DT_RIGHT	Right-justify the text.
DT_VCENTER	Vertically center the text.

DT_TOP	Top-justify the text.
DT_BOTTOM	Bottom-justify the text.
DT_HALFTONE	Halftone the text display.
DT_MNEMONIC	If a mnemonic prefix character is encountered, the next character is drawn with mnemonic emphasis.
DT_QUERYEXTENT	No drawing is performed. **Rectangle** is changed to a rectangle that bounds the string if it were drawn with WinDrawText.
DT_WORDBREAK	Only words that fit completely within the supplied rectangle are drawn. A *word* is defined as: Any number of leading spaces followed by one or more visible characters and terminated by a space, carriage return, or line-feed character. When calculating whether a particular word fits within the given rectangle, this function does not consider the trailing blanks. Only the length of the visible part of the word is tested against the right edge of the rectangle. Also, note that this function always tries to draw at least one word, even if that word does not fit in the passed rectangle. This is so that progress is always made when drawing multi-line text.
DT_EXTERNALLEADING	This flag causes the "external leading" value for the current font to be added to the bottom of the bounding rectangle before returning. It only has any effect when both DT_TOP and DT_QUERYEXTENT are also specified.
DT_TEXTATTRS	If this is specified, text is drawn using the character foreground and background colors of the presentation space, and **ForeColor** and **BackColor** are ignored.
DT_ERASERECT	If this is specified, the rectangle defined by **Rectangle** is erased before drawing the text. Otherwise, the background of the characters themselves can be erased if the character background mix (see GpiSetAttrs and GpiSetBackMix) is set to BM_OVERPAINT.

Chars *(SHORT)* — return

Count of characters drawn within the rectangle.

If DT_WORDBREAK is specified, this parameter returns the number of characters displayed. However, if the first word of the string does not fit in the rectangle, this parameter reflects the fact that the entire word is drawn.

If DT_WORDBREAK is **not** specified, the count returned is the full length of the string regardless of how much fits into the bounding rectangle.

0	Error occurred
Other	Count of characters drawn within the rectangle.

Possible returns from WinGetLastError:

PMERR_INVALID_FLAG

Remarks

Text is always drawn in the current font with the current foreground and background mix modes.

This call must only be used in draw mode (DM_DRAW), to a screen device context; **hps** can be either a micro-presentation space or a normal presentation space (see GpiCreatePS).

WinEmptyClipbrd — Empty Clipboard

This call empties the clipboard, removing and freeing all handles to data that is in the clipboard.

```
WinEmptyClipbrd (hab, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The clipboard must be opened using WinOpenClipbrd before using this call.

This call requires the existence of a message queue.

WinEnablePhysInput – Enable Physical Input

This call enables or disables queuing of physical input.

WinEnablePhysInput (DeskTop, NewInputState, *OldInputState***)**

Parameters

DeskTop *(HWND)* – input
Desktop-window handle:

HWND_DESKTOP	The desktop window handle
Other	Specified desktop window handle.

NewInputState *(BOOL)* – input
New state for the queuing of physical input:

TRUE	Pointing device and keyboard input are queued
FALSE	Pointing device and keyboard input are disabled.

OldInputState *(BOOL)* – return
Previous state for the queuing of physical input:

TRUE	Pointing device and keyboard input were queued
FALSE	Pointing device and keyboard input were disabled.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinEnableWindow — Set Window Enabled State

SAA

This call sets the window enabled state.

```
WinEnableWindow (hwnd, NewEnabled, Success)
```

Parameters

hwnd *(HWND)* — input
Window handle.

NewEnabled *(BOOL)* — input
New enabled state:

TRUE Set window state to enabled
FALSE Set window state to disabled.

Success *(BOOL)* — return
Window enabled indicator:

TRUE Window enabled state successfully updated
FALSE Window enabled state not successfully updated.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

If the enable state of **hwnd** is changing, a WM_ENABLE message is sent before this call returns.

If a window is disabled, all its children are implicitly disabled, although they are not sent the WM_ENABLE message. Typically, a window changes appearance when disabled. For example, a disabled pushbutton is displayed with halftone text.

If **hwnd** is disabled, and it, or one of its descendants, is the focus window, that window loses the focus, so that no window has the focus. However, a disabled window may subsequently be assigned the focus, in which instance it should respond to keyboard input.

WinEnableWindowUpdate — Enable Window Update

This call sets the window visibility state for subsequent drawing, without causing any change to the window on the device.

WinEnableWindowUpdate (hwnd, NewVisibility, *Success***)**

Parameters

hwnd *(HWND)* — input
Window handle.

NewVisibility *(BOOL)* — input
New visibility state:

TRUE Set window state visible
FALSE Set window state invisible.

Success *(BOOL)* — return
Visibility-changed indicator:

TRUE Window visibility successfully changed
FALSE Window visibility not successfully changed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call can be used to defer drawing when making a series of changes to the window. The window can be redrawn by use of the WinShowWindow call.

WS_VISIBLE is set to **NewVisibility** without causing redrawing. This implies that if the window was previously visible, it remains visible on the device when WS_VISIBLE is reset, and conversely, if the window was previously invisible, it is not shown when WS_VISIBLE is set. If **NewVisibility** is set to TRUE, any subsequent drawing into the window is visible. If **NewVisibility** is set to FALSE, any subsequent drawing into the window is not visible.

If the value of the WS_VISIBLE style bit has been changed, the WM_SHOW message is sent to the window of **hwnd** before the call returns. This call is usually used to disable drawing before making a series of changes to a window to prevent unnecessary drawing. To show a window and ensure that it is redrawn after calling the WinEnableWindowUpdate call with **NewVisibility** set to FALSE, use the WinShowWindow call with **NewVisibility** set to TRUE.

Any alteration to the appearance of a window disabled for window update is not presented. Therefore, the application must ensure that the window is redrawn. In particular, if a window is destroyed while in this state its image is not removed from the display. After window updating is re-enabled, the application should ensure that the window gets totally invalidated so that it repaints.

WinEndEnumWindows — End Window Enumeration

This call ends the enumeration process for a specified enumeration.

```
WinEndEnumWindows (henum, Success)
```

Parameters

henum *(HENUM)* — input
Enumeration handle.

Returned by previous call to the WinBeginEnumWindows call.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HENUM

Remarks

This call destroys the window hierarchy remembered by the WinBeginEnumWindows call. After this call, the **henum** parameter is no longer valid.

WinEndPaint — End Paint

This call indicates that the redrawing of a window is complete, generally as part of the processing of a WM_PAINT message.

WinEndPaint (hps, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

Handle of the presentation space that is used for drawing, following a WinBeginPaint call.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The presentation space is restored to its state before the WinBeginPaint call:

- Cache presentation space is returned to the cache.
- Other presentation spaces have their original drawing state restored, including reassociating the original device context (if there was one).

If the pointer is hidden by the WinBeginPaint call, it is reshown by this call.

Any child windows having a synchronous painting style of the window associated with the presentation space are updated during the processing of this call, if they have non-NULL update regions.

This call requires the existence of a message queue.

WinEnumClipbrdFmts — Enumerate Clipboard Formats

This call enumerates the list of clipboard data formats available in the clipboard.

```
WinEnumClipbrdFmts (hab, Prev, Next)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Prev *(USHORT)* — input
Previous clipboard-data format index.

Specifies the index of the last clipboard data format enumerated using this call.

This should start at zero, in which instance the first available format is obtained. Subsequently, it should be set to the last format index value returned by this call.

Next *(USHORT)* — return
Next clipboard-data format index:

0 Enumeration is complete; that is, there are no more clipboard formats available.
Other Index of the next available clipboard-data format in the clipboard.

Remarks

The clipboard should be open before this call is used.

This call requires the existence of a message queue.

WinEnumDlgItem — Enumerate Dialog Item

This call returns the window handle of a dialog item within a dialog window.

WinEnumDlgItem (Dlg, hwnd, Code, Lock, *Item*)

Parameters

Dlg *(HWND)* — input
Dialog-window handle.

hwnd *(HWND)* — input
Child-window handle.

This may be an immediate child of the dialog window or a window lower in the window hierarchy, such as a child of a child window.

Code *(USHORT)* — input
Item-type code.

Determines the type of dialog item to return.

EDI_PREVTABITEM	Previous item with style WS_TABSTOP. Wraps around to end of dialog item list when beginning is reached.
EDI_NEXTTABITEM	Next item with style WS_TABSTOP. Wraps around to beginning of dialog item list when end is reached.
EDI_FIRSTTABITEM	First item in dialog with style WS_TABSTOP. **hwnd** is ignored.
EDI_LASTTABITEM	Last item in dialog with style WS_TABSTOP. **hwnd** is ignored.
EDI_PREVGROUPITEM	Previous item in the same group. Wraps around to end of group when the start of the group is reached. For information on the WS_GROUP style, see "Window Styles" on page 12-2.
EDI_NEXTGROUPITEM	Next item in the same group. Wraps around to beginning of group when the end of the group is reached.
EDI_FIRSTGROUPITEM	First item in the same group.
EDI_LASTGROUPITEM	Last item in the same group.

Lock *(BOOL)* — input
Lock indicator.

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

Item *(HWND)* — return
Item-window handle.

As dictated by **Code**.

The window is always an immediate child of **Dlg**, even if **hwnd** is not an immediate child window.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinEqualRect — Equal Rectangle

This call compares two rectangles for equality.

```
WinEqualRect (hab, Rect1, Rect2, Equal)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Rect1 *(RECT)* — input
First rectangle.

Note: The data type *WRECT* may also be used, if supported by the language.

Rect2 *(RECT)* — input
Second rectangle.

Note: The data type *WRECT* may also be used, if supported by the language.

Equal *(BOOL)* — return
Equality indicator:

TRUE Rectangles are identical
FALSE Rectangles are not identical, or an error occurred.

WinExcludeUpdateRegion — Exclude Update Region

This call subtracts the update region (invalid region) of a window from the clipping region of a presentation space.

WinExcludeUpdateRegion (hps, hwnd, *Complexity*)

Parameters

hps *(HPS)* — input
Presentation-space handle whose clipping region is to be updated.

hwnd *(HWND)* — input
Window handle.

Handle of window whose update region is subtracted from the clipping region of the presentation space.

Complexity *(SHORT)* — return
Complexity value.

This indicates the resulting form of the clipping area. The values and meanings of this parameter are defined in the GpiCombineRegion function.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is typically used to prevent drawing into parts of a window that are known to be invalid, as during an incremental update optimization process.

It is the application's responsibility to reset the clipping region when necessary.

This call requires the existence of a message queue.

WinFillRect — Fill Rectangle

SAA

This call draws a filled rectangular area.

WinFillRect (hps, Rect, Color, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

This can be either a micro-presentation space or a normal presentation space.

Rect *(RECT)* — input
Rectangle to be filled, in window coordinates.

Points on the left and bottom boundaries of the rectangle are included in the fill, but points on the right and top boundaries are not, except where they are also on the left and bottom boundaries; that is, the top-left and bottom-right corners.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

Color *(LONG)* — input
Color with which to fill the rectangle.

This is either a color index, or an RGB color value, depending upon whether and how a logical color table has been loaded. (See the GpiCreateLogColorTable and GpiSetColor functions.)

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call does not change any presentation space state.

This call must only be used in draw mode (DM_DRAW) to a screen device context.

If an empty rectangle is specified, this call draws nothing and completes successfully (that is, TRUE is returned).

WinFindAtom — Find Atom

This call finds an atom in the atom table.

WinFindAtom (AtomTbl, AtomName, *atom*)

Parameters

AtomTbl *(HATOMTBL)* — input
Atom-table handle.

This is the handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable call.

AtomName *(STRL)* — input
Atom name.

This is a null terminated character string to be found in the table.

If the string begins with a "#" character, then the five ASCII digits that follow are converted into an integer atom.

If the string begins with a "!" character, then the next two bytes are interpreted as an atom.

If the high order word of the string is minus one, then the low order word is an atom.

atom *(ATOM)* — return
Atom value:

Atom The atom associated with the passed string
0 Invalid atom table handle or invalid atom name specified.

Possible returns from WinGetLastError:

PMERR_INVALID_HATOMTBL
PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND

Remarks

This call is identical to the WinAddAtom call, except that:

- If the atom name is not found in the table, it is not added to the table and 0 is returned.
- If the atom name is found in the table, the use count is not incremented.

Since integer atoms do not have a use count and do not actually occupy memory in the atom table, this call is identical to WinAddAtom with respect to integer atoms.

WinFlashWindow — Flash Window

SAA

This call starts or stops a window flashing.

```
WinFlashWindow (hwnd, Flash, Success)
```

Parameters

hwnd *(HWND)* — input
Handle of window to be flashed.

Flash *(BOOL)* — input
Start-flashing indicator:

TRUE Start window flashing
FALSE Stop window flashing.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

Flashing a window brings the user's attention to a window that is not the active window, where some important message or dialog must be seen by the user.

Flashing is typically done by inverting the title bar continuously. The alarm is sounded for the first five flashes.

Programming Note: It should be used only for important messages, for example, where some component of the system is failing and requires immediate attention to avoid damage.

This call requires the existence of a message queue.

WinFocusChange — Change Focus Window

This call changes the focus window.

WinFocusChange (DeskTop, NewFocus, FocusChange, *Success***)**

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

NewFocus *(HWND)* — input
Window handle to receive the focus.

FocusChange *(BIT16)* — input
Focus changing indicators.

These indicators are passed on in the WM_FOCUSCHANGE message:

FC_NOSETFOCUS
Do not send the WM_SETFOCUS message to the window receiving the focus.
FC_NOLOSEFOCUS
Do not send the WM_SETFOCUS message to the window losing the focus.
FC_NOSETACTIVE
Do not send the WM_ACTIVATE message to the window being activated.
FC_NOLOSEACTIVE
Do not send the WM_ACTIVATE message to the window being deactivated.
FC_NOSETSELECTION
Do not send the WM_SETSELECTION message to the window being selected.
FC_NOLOSESELECTION
Do not send the WM_SETSELECTION message to the window being deselected.
FC_NOBRINGTOTOP
Do not bring any window to the top.
FC_NOBRINGTOTOPFIRSTWINDOW
Do not bring the first frame window to the top.
FC_SETACTIVEFOCUS
Set the focus to the child window that previously had the focus of the first window in the parentage of **NewFocus**, which has the CS_FRAME style.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call sends a WM_FOCUSCHANGE message to the window that is losing the focus and a WM_FOCUSCHANGE message to the window that is receiving the focus.

This function fails if another process or thread is currently using this function.

Other messages may be sent as a consequence of the frame control processing of the WM_FOCUSCHANGE message, depending on the value of the **FocusChange** parameter. These messages, if sent, are sent in this order:

1. WM_SETFOCUS to the window losing the focus.
2. WM_SETSELECTION to the windows losing their selection.
3. WM_ACTIVATE to the windows being deactivated.
4. WM_ACTIVATE to the windows being activated.
5. WM_SETSELECTION to the windows being selected.
6. WM_SETFOCUS to the window receiving the focus.

Programming Note: Use of the WinQueryFocus call during processing of this call results in the window handle of the window losing the focus being returned while the WM_FOCUSCHANGE message with the **setfocus** parameter set to FALSE is being processed, and the window handle of the window receiving the focus being returned while the WM_FOCUSCHANGE message with the **setfocus** parameter set to TRUE is being processed.

Use of the WinQueryActiveWindow call during processing of this call results in the window handle of the window being deactivated being returned while the WM_ACTIVATE message with the **active** parameter set to FALSE is being processed, and the window handle of the window being activated being returned while the WM_ACTIVATE message with the **active** parameter set to TRUE is being processed.

Also, there is a short period during the time after the old active window has acted on the deactivation message and before the new active window has acted on the activation message when the WinQueryActiveWindow call returns NULL.

This call should not be made unless it is directly or indirectly the result of operator input.

Even if FC_NOSETSELECTION is not specified, the WM_SETSELECTION is not sent to a frame window that is already selected. This can occur if the focus is being transferred from a parent to a child window and FC_NOLOSESELECTION was specified.

This call requires the existence of a message queue.

WinFreeErrorInfo — Free Error Information

This call releases memory allocated for an error-information block.

WinFreeErrorInfo (ErrorInfo, *Success***)**

Parameters

ErrorInfo *(ERRINFO)* — input
Error-information block whose memory is to be released.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE **ErrorInfo** is not an error-information block for the current thread.

Remarks

This call requires the existence of a message queue.

WinFreeMem — Free Memory on Heap

This call frees memory, allocated by the WinAllocMem call.

```
WinFreeMem (Heap, Mem, Length, Return)
```

Parameters

Heap *(HHEAP)* — input
Handle to a heap.

This must have been returned by a previous call to WinCreateHeap.

Mem *(SEGOFF)* — input
Memory block to be freed.

This must have been returned by a previous call to WinAllocMem or WinReallocMem.

Length *(USHORT)* — input
Size of the memory to be freed.

Must match the allocated size of the block.

Return *(SEGOFF)* — return
Values as follows:

NULL Successful completion
Other The **Mem** parameter.

Remarks

Except for the two low bits, which are ignored, the value of the **Mem** parameter must have been returned by either the WinAllocMem or WinReallocMem call.

If the passed heap is created with the HM_MOVEABLE option, the value of the **Mem** parameter is ignored and the value of the size word in the allocated object is used instead. If the handle word is non-zero, the contents of the handle value word are set to zero after the object has been freed.

This call 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 call does NOT attempt to coalesce the block being freed with other free blocks. Use the WinAvailMem call to force free blocks to be coalesced.

WinFreeMsg — Free Message

This call destroys the data structures associated with the message specified.

WinFreeMsg (hab, hwnd, Msgid, Param1, Param2, Reply, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

hwnd *(HWND)* — input
Window handle.

This is the handle of the window associated with the message.

Msgid *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input/output
Message parameter 1.

If specified as NULL, no action is taken for this message parameter.

PM always returns a value of NULL.

Param2 *(MPARAM)* — input/output
Message parameter 2.

If specified as NULL, no action is taken for this message parameter.

PM always returns a value of NULL.

Reply *(MRESULT)* — input/output
Message reply.

If specified as NULL, no action is taken for this message parameter.

PM always returns a value of NULL.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB

Remarks

This call should be used when the application is satisfied that the data structures associated with the message parameters or message reply are no longer required. Failure to use this call for each message received by WinGetMsg or WinPeekMsg, or for each message reply received from WinDispatchMsg, WinDefWindowProc and similar calls, may result in storage errors.

This call may be used for all system messages, without regard for the nature of the message parameters. If used for user messages, the message parameters and message reply are assumed to be BIT32.

WinFreeMsg — Free Message

SAA

The call has the effect of issuing the WinDestroyDataStructure call for each of the message parameters or message reply that is a handle to a data structure. Using WinFreeMsg relieves the application of the need to understand the nature of the message parameters or message reply.

Programming Note: This function is required only in COBOL and FORTRAN.

WinGetClipPS – Get Clipped Presentation Space

This call obtains a clipped cache presentation space.

WinGetClipPS (hwnd, ClipWindow, Clipflags, *hps*)

Parameters

hwnd *(HWND)* – input
Handle of window for which the presentation space is required.

ClipWindow *(HWND)* – input
Handle of window for clipping.

Clipflags *(USHORT)* – input
Clipping control flags.

PSF_CLIPSIBLINGS	Clip out all siblings of **hwnd**.
PSF_CLIPCHILDREN	Clip out all children of **hwnd**.
PSF_CLIPUPWARDS	Taking **ClipWindow** as a reference window clip out all sibling windows before **ClipWindow**. This value may not be used with PSF_CLIPDOWNWARDS.
PSF_CLIPDOWNWARDS	Taking **ClipWindow** as a reference window clip out all sibling windows after **ClipWindow**. This value may not be used with PSF_CLIPUPWARDS.
PSF_LOCKWINDOWUPDATE	Calculate a presentation space that keeps a visible region even though output may be locked by the WinLockWindowUpdate function.
PSF_PARENTCLIP	Calculate a presentation space that uses the visible region of the parent of **hwnd** but with an origin calculated for **hwnd**.

hps *(HPS)* – return
Presentation-space handle that can be used for drawing.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The presentation space obtained by this call is a cache "micro-presentation space" present in the system. This can be used for simple drawing operations that do not depend on long-term data being stored in the presentation space.

This call requires the existence of a message queue.

WinGetCurrentTime – Get Current Time

This call returns the current time.

WinGetCurrentTime (hab, *Time*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

Time *(ULONG)* – return
System-timer count.

The time is in milliseconds, from the system Initial Program Load (IPL). This is the same value as stored in the information segment.

WinGetDlgMsg — Get Dialog Message

This call obtains a message from the application's queue associated with the specified dialog.

WinGetDlgMsg (Dlg, msg, *Result*)

Parameters

Dlg *(HWND)* — input
Dialog-window handle.

msg *(QMSG)* — output
Message structure.

Result *(BOOL)* — return
Continue message indicator:

TRUE Message returned is not a WM_QUIT message and the dialog has not been dismissed.
FALSE Message returned is a WM_QUIT message or the dialog has been dismissed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call enables a language which cannot support window procedures to provide the function of a modal dialog. The application creates a modeless dialog by the use of the WinCreateDlg or the WinLoadDlg calls and then issues this call in order to process messages only associated with the dialog.

The first time that this call is issued, the owner of the window specified by **Dlg** is disabled, thereby preventing input into windows other than the dialog. The owner of the window specified by **Dlg** is enabled when the WinDismissDlg call is issued either by the application or by the default dialog procedure.

If a WM_QUIT is encountered, WinGetDlgMsg itself issues a WinDismissDlg call, and posts the WM_QUIT message back to the queue so that the application main loop terminates in the normal way.

This call requires the existence of a message queue.

WinGetErrorInfo — Get Error Information

SAA

This call returns detailed error information.

WinGetErrorInfo (hab, *ErrorInfo***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

ErrorInfo *(ERRINFO)* — return
Error information.

This structure contains information about the previous error code for the current thread:

NULL No error information available
Other Error information.

Remarks

This call allocates a single private segment to contain the *ERRINFO* structure. All of the pointers to string fields within the *ERRINFO* structure are offsets to memory within that segment.

The memory allocated by this function is not released until the returned pointer is passed to the WinFreeErrorInfo call.

Like the WinGetLastError call, this call releases any saved error information after formatting the error message. If this call is repeated consecutively, the second call returns NULL, as the saved error information is consumed by the first call.

This call requires the existence of a message queue.

WinGetKeyState — Get Key State

This call returns the state of the key at the time that the last message obtained from the queue was posted.

WinGetKeyState (DeskTop, Vk, *KeyState*)

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Vk *(SHORT)* — input
Virtual key value.

Contains the virtual key value in the low-order byte, and zero in the high-order byte.

KeyState *(SHORT)* — return
Key state.

This value is the "OR" combination of the following bits:

X'0001'	The key has been pressed an odd number of times since the system has been started.
X'8000'	The key is down.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

See also the WinGetPhysKeyState call. This call is used to determine whether a virtual key is up, down, or toggled.

This call can be used to obtain the state of the pointing device buttons with the VK_BUTTON1, VK_BUTTON2, and VK_BUTTON3 virtual key codes.

This call requires the existence of a message queue.

WinGetLastError — Get Last Error

SAA

This call returns the error state set by the failure of a Presentation Manager function.

WinGetLastError (hab, *ErrorCode***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

ErrorCode *(ERRORID)* — return
Last-error state.

Remarks

Returns the last nonzero error code, and sets the error code to zero.

The current error state is reset to zero.

In multiple thread applications where there are multiple anchor blocks, errors are stored in the anchor block created by the WinInitialize call of the thread invoking a call. The application is responsible for specifying the correct anchor block for the thread issuing this call.

This call requires the existence of a message queue.

WinGetMinPosition — Get Minimum Position

This call returns the position to which a window is minimized.

WinGetMinPosition (hwnd, Swp, Point, *Success***)**

Parameters

hwnd *(HWND)* — input
Frame-window handle.

Swp *(SWP)* — output
Set window position structure.

The SWP_SIZE and SWP_MOVE indicators are set in this parameter on return from this call, implying that the **x**, **y**, **width**, and **height** parameters have been initialized.

Point *(POINT)* — input
Preferred position:

NULL System is to choose the position
Other System is to choose the position nearest to the specified point.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion.

The WS_MINIMIZE style is set for **hwnd**. This enables the system to determine which other frame windows are minimized, during the enumeration process performed by this function.

Also, the window words QWS_XMINIMIZE and QWS_YMINIMIZE for **hwnd** are initialized. This enables the system to ensure that no windows that have been, or are being, minimized use the same position.

FALSE Error occurred.

Remarks

This call chooses the position for a minimized window. It enumerates all the siblings of the specified window to determine the first available position.

This call requires the existence of a message queue.

WinGetMsg — Get Message

SAA

This call gets, waiting if necessary, a message from the thread's message queue and returns **msg** when a message conforming to the filtering criteria is available.

```
WinGetMsg (hab, msg, Filter, First, Last, Result)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

msg *(QMSG)* — output
Message structure.

Filter *(HWND)* — input
Window filter.

First *(USHORT)* — input
First message identity.

Last *(USHORT)* — input
Last message identity.

Result *(BOOL)* — return
Continue message indicator:

TRUE Message returned is not a WM_QUIT message
FALSE Message returned is a WM_QUIT message.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

If system or queue hooks are installed, they are called before this function returns.

Result is generally used to determine when to terminate the application's main loop and exit the program.

Filter constrains the returned message to be for a specific window or its children. When **Filter** is null, the returned message can be for any window. The message identity is restricted to the range of message identities specified by **First** and **Last** inclusive. When **First** and **Last** are both zero, any message satisfies the range constraint. When **First** is greater than **Last**, messages, except those whose identities lie between **First** and **Last**, are eligible to be returned. Messages that do not conform to the filtering criteria remain in the queue.

When **Filter** is null, and **First** and **Last** are both zero, all messages are returned in the order that they were posted to the queue.

By using filtering, messages can be processed in an order that is different from the one in the queue. Filtering is used in situations where the application wants to receive messages of a particular type, rather than having to deal with other types of message at an inconvenient point in the logic of the application. For example, when a "mouse down" message is received, filtering can be used to wait for the "mouse up" message without having to be concerned with receiving other messages.

These constants can also be used when filtering messages:

WM_MOUSEFIRST	Lowest value pointing device message
WM_MOUSELAST	Highest value pointing device message
WM_BUTTONCLICKFIRST	Lowest value pointing device button click message
WM_BUTTONCLICKLAST	Highest value pointing device button click message
WM_DDE_FIRST	Lowest value DDE message
WM_DDE_LAST	Highest value DDE message.

Great care must be taken if filtering is used, to ensure that a message that satisfies the specification of the filtering parameters can occur, otherwise this call cannot complete. For example, calling this function with **First** and **Last** equal to WM_CHAR and with **Filter** set to a window handle that does not have the input focus, prevents this function from returning.

Key strokes are passed to the WinTranslateAccel call, which implies that accelerator keys are translated into WM_COMMAND or WM_SYSCOMMAND messages, and so are not seen as WM_CHAR messages by the application.

Programming Note: An application must be prepared to receive messages other than those documented in this publication. All messages that an application does not want to handle should be sent to WinDefWindowProc.

This call requires the existence of a message queue.

WinGetNextWindow — Get Next Window

This call gets the window handle of the next window in a specified enumeration list.

```
WinGetNextWindow (henum, Next)
```

Parameters

henum *(HENUM)* — input
Enumeration handle.

Returned by previous call to the WinBeginEnumWindows call.

Next *(HWND)* — return
Next window handle in enumeration list:

NULL Error occurred, **henum** was NULL, or all the windows have been enumerated
Other Next window handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HENUM

Remarks

Enumeration starts with the topmost child window and then proceeds downwards through the enumeration list each time the call is called, until all the windows have been enumerated. At this point, the call returns NULL. The enumeration then wraps and the handle of the topmost child window is returned on the next call. This call does not lock windows. Window locking is not required in OS/2 Version 1.2 and above.

This call requires the existence of a message queue.

WinGetPhysKeyState – Get Physical Key State

This call returns the physical key state.

WinGetPhysKeyState (DeskTop, Scancode, *KeyState***)**

Parameters

DeskTop *(HWND)* – input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Scancode *(SHORT)* – input
Hardware scancode.

Contains the scancode value in the low-order byte, and zero in the high-order byte.

KeyState *(SHORT)* – return
Key state:

This value is the "OR" combination of the following bits:

X'0001'	The key has been pressed an odd number of times since the system has been started.
X'0002'	The key has been pressed since the last time this call was issued, or since the system has been started if this is the first time the call has been issued.
X'8000'	The key is down.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call returns information about the asynchronous (interrupt level) state of the virtual key indicated by the **Scancode** parameter.

This call returns the physical state of the key; it is not synchronized to the processing of input. (See the WinGetKeyState call.)

This call requires the existence of a message queue.

WinGetPS – Get Presentation Space

SAA

This call gets a cache presentation space.

WinGetPS (hwnd, *hps*)

Parameters

hwnd *(HWND)* – input
Handle of window for which the presentation space is required:

HWND_DESKTOP	The desktop-window handle; a presentation space for the whole of the desktop window is returned.
Other	Handle of window for which the presentation space is required.

hps *(HPS)* – return
Presentation-space handle that can be used for drawing in the window.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The presentation space got by this call is a cache "micro-presentation space" present in the system. This can be used for simple drawing operations that do not depend on long-term data being stored in the presentation space.

The initial state of the presentation space is the same as that of a presentation space created using the GpiCreatePS call. The color table is in default color index mode. The visible region associated with **hps** depends upon the window and class styles of **hwnd**:

Style	**Visible region of presentation space**
WS_CLIPCHILDREN	All the window's child windows are excluded.
WS_CLIPSIBLINGS	All the sibling windows of **hwnd** are excluded.
CS_PARENTCLIP	Is the same as that of the window's parent window. The presentation space origin is established normally; that is, relative to the lower left of the window itself, not its parent. This style optimizes the use of the presentation space cache by minimizing the calculation of the visible region for child windows.

WinGetScreenPS – Get Screen Presentation Space

This call returns a presentation space which can be used for drawing anywhere on the screen.

```
WinGetScreenPS (DeskTop, ScreenPS)
```

Parameters

DeskTop *(HWND)* – input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

ScreenPS *(HPS)* – return
Presentation-space handle.

A micro-presentation space which can be used for drawing over the entire desktop window (the whole screen):

NULL	**DeskTop** is not HWND_DESKTOP or a desktop window handle obtained from the WinQueryDesktopWindow call.
Other	Presentation-space handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

Take great care when using this call. The returned presentation space is not clipped to any of the other windows present on the screen. Thus it is possible to draw in regions belonging to windows of other threads and processes.

The WinLockWindowUpdate call should be used to avoid simultaneous updates to the same part of the screen. This does not cause the presentation space returned by this call to become clipped in any way. Care of the appearance of windows of other threads is still the responsibility of the user of the screen presentation space.

When the application finishes using the screen presentation space, it should be destroyed using the WinReleasePS call.

This call requires the existence of a message queue.

WinGetSysBitmap — Get System Bit Map

SAA

This call returns a handle to one of the standard bit maps provided by the system.

```
WinGetSysBitmap (DeskTop, Index, hbm)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Index *(INDEX2)* — input
System bit-map index value:

SBMP_SYSMENU	System menu
SBMP_SBUPARROW	Scroll bar up arrow
SBMP_SBDNARROW	Scroll bar down arrow
SBMP_SBRGARROW	Scroll bar right arrow
SBMP_SBLFARROW	Scroll bar left arrow
SBMP_MENUCHECK	Menu check mark
SBMP_CHECKBOXES	Checkbox/radio button check mark
SBMP_BTNCORNERS	Pushbutton corners
SBMP_MINBUTTON	Minimize button
SBMP_MAXBUTTON	Maximize button
SBMP_RESTOREBUTTON	Restore button
SBMP_SBUPDBLARROW	Scroll bar up double arrow
SBMP_SBDNDBLARROW	Scroll bar down double arrow
SBMP_SBRGDBLARROW	Scroll bar right double arrow
SBMP_SBLFDBLARROW	Scroll bar left double arrow
SBMP_INFORMATION	Black information 'i' in a square box
SBMP_QUERY	Question mark in a square box
SBMP_WARNING	Black '!' in a square box
SBMP_ERROR	STOP sign on a white background.

hbm *(HBITMAP)* — return
System bit-map handle.

NULL	Error occurred
Other	System bit-map handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE
PMERR_RESOURCE_NOT_FOUND

Remarks

The bit map returned can be used for any of the normal bit-map operations. This call provides a new copy of the system bit map each time it is called. The application should release any bit maps it gets with this call by using the GpiDeleteBitmap call.

This call requires the existence of a message queue.

WinInflateRect – Inflate Rectangle

This call expands a rectangle.

```
WinInflateRect (hab, rect, cx, cy, Success)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

rect *(RECT)* – input/output
Rectangle to be expanded.

Note: The data type *WRECT* may also be used, if supported by the language.

cx *(SHORT)* – input
Horizontal expansion.

cy *(SHORT)* – input
Vertical expansion.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call adjusts the size of the rectangle by applying the **cx** parameter horizontally at both vertical edges and the **cy** parameter vertically at both horizontal edges.

The **cx** parameter is subtracted from the left and added to the right of the rectangle, and the **cy** parameter is subtracted from the bottom and added to the top of the rectangle.

If the values of the **cx** and **cy** parameters are both positive, the rectangle is enlarged and surrounds the original rectangle. Conversely, if both these values are negative, the rectangle is reduced in size and is inset with respect to the original rectangle.

WinInitialize – Initialize

SAA

This call initializes the Presentation Manager facilities for use by an application.

```
WinInitialize (Options, hab)
```

Parameters

Options *(BIT16)* – input
Initialization options:

NULL The initial state for newly created windows is that all messages for the window are available for processing by the application.

This is the only option available in the Presentation Manager.

hab *(HAB)* – return
Anchor-block handle:

NULL An error occurred. For example, this call is issued twice on the same thread without an intervening WinTerminate call.
Other Anchor-block handle.

Remarks

This must be the first Presentation Manager call issued by any application thread using Presentation Manager facilities.

It returns **hab**, which is NULL if the initialization is not successful.

IBM Operating System/2 does not generally use the information supplied by the **hab** parameter to its calls; instead, it deduces it from the identity of the thread that is making the call. Thus an IBM Operating System/2 application is not required to supply any particular value as the **hab** parameter. However, in order to be portable to other environments, an application must provide the **hab**, that is returned by the WinInitialize call of the thread, to any IBM Operating System/2 call that requires it.

Options determines the initial state of message processing with respect to a created window.

This call requires the existence of a message queue.

WinInSendMsg – In Send Message

This call determines whether the current thread is processing a message sent by another thread.

WinInSendMsg (hab, *Processing*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

Processing *(BOOL)* – return
Message-processing indicator:

TRUE Current thread is processing a message sent by another thread
FALSE Current thread is not processing a message or an error occurred.

Remarks

If the message is from another thread this call determines whether or not the message was initiated by the active thread. The 'active thread' is the thread associated with the current active window. (See also the WinIsThreadActive call.)

Typically this call is used by applications to determine how to proceed with errors when the window processing the message is not the active window. For example, if the active window uses the WinSendMsg call 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 methods an inactive window has to inform the user of an error, are to create a message box (see WinMessageBox), or to flash a window (see WinFlashWindow).

WinInstStartApp — Start Installed Application

This call starts a program from those installed in the program list.

```
WinInstStartApp (hini, NotifyWindow, Count, Application, CmdLine, Data, Options, happ)
```

Parameters

hini *(HINI)* — input
Initialization-file handle.

NotifyWindow *(HWND)* — input
Notification-window handle.

A WM_APPTERMINATENOTIFY message is posted to this window, when the started application terminates.

NULL Do not post the notification message
Other Post the notification message to this window.

Count *(COUNT2)* — input
Count of elements in the **Application** parameter. Must be 1 or 2.

Application *(STRL*Count)* — input
Identifier of the application to be started.

The first element of this array contains the name of the application as known in the program list. If a second element is present, it is the group name needed to identify this application uniquely. If no group name element is specified, the default group is assumed.

CmdLine *(STRL)* — input
Input parameters for the application to be started.

This string specifies the command line parameters to be passed to this application when it starts.

NULL There are no parameters to be passed to the application
Other The parameters to be passed to the application.

Data *(STORAGE)* — input
Start data.

Reserved, must be NULL.

Options *(BIT16)* — input
Option indicators.

If more than one option is selected, the values can be ORed together.

NULL	No options selected.
SAF_INSTALLEDCMDLINE	The command line parameters installed in the program starter list are used; the **CmdLine** parameter is ignored.
SAF_STARTCHILDAPP	The specified application is started as a child session of the session from which WinInstStartApp is issued. The calling application may terminate the called application with a WinTerminateApp call.

happ *(HAPP)* — return
Application handle.

NULL Application not started
Other Application handle.

Possible returns from WinGetLastError:

PMERR_INVALID_PARAMETERS
PMERR_INVALID_APPL
PMERR_INVALID_WINDOW
PMERR_CANNOT_START
PMERR_STARTED_IN_BACKGROUND (warning
PMERR_DOS_ERROR
PMERR_NOT_ENOUGH_MEM

Remarks

Starts the application identified by the **Application** parameter. The **hini** parameter specifies the profile containing the program list.

If the application is successfully started, an application handle is returned in the **happ** parameter. If SAF_STARTCHILDAPP is specified, this can be used to stop the application (see the WinTerminateApp call).

When the program specified by the **happ** parameter terminates, the window specified by the **NotifyWindow** parameter (if the window still exists and is valid) has a WM_APPTERMINATENOTIFY message posted to it to notify it of the application termination.

This call requires the existence of a message queue.

WinIntersectRect – Intersect Rectangle

This call calculates the intersection of the two source rectangles and returns the result in the destination rectangle.

```
WinIntersectRect (hab, Dest, Rect1, Rect2, Success)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

Dest *(RECT)* – output
Intersection rectangle.

Is the intersection of **Rect1** and **Rect2**.

Note: The data type *WRECT* may also be used, if supported by the language.

Rect1 *(RECT)* – input
First rectangle.

Note: The data type *WRECT* may also be used, if supported by the language.

Rect2 *(RECT)* – input
Second rectangle.

Note: The data type *WRECT* may also be used, if supported by the language.

Success *(BOOL)* – return
Success indicator:

TRUE Source rectangles intersect
FALSE Source rectangles do not intersect, or an error occurred.

Remarks

If there is no intersection, an empty rectangle is returned in **Dest**.

WinInvalidateRect — Invalidate Rectangle

This call adds a rectangle to a window's update region.

WinInvalidateRect (hwnd, Prc, IncludeClippedChildren, *Success***)**

Parameters

hwnd *(HWND)* — input
Handle of window whose update region is to be changed:

HWND_DESKTOP	This call applies to the whole screen (or desktop)
Other	Handle of window whose update region is to be changed.

Prc *(RECT)* — input
Update rectangle.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

NULL	The whole window is to be added into the window's update region.
Other	Rectangle to be added to the window's update region.

IncludeClippedChildren *(BOOL)* — input
Invalidation-scope indicator:

TRUE	Include the descendants of **hwnd** in the invalid rectangle
FALSE	Include the descendants of **hwnd** in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The update region is a subregion of a window that is deemed "invalid" or incorrect in visual terms and in need of redrawing.

If the window has a CS_SYNCPAINT style, it is redrawn during the processing of this call and the update region should be NULL on return from this call.

If the window has a WS_CLIPCHILDREN style with part of its update region overlapping child windows with a CS_SYNCPAINT style, those children are updated before this call returns.

This function should not be called in response to a WM_PAINT request for windows of style CS_SYNCPAINT. CS_SYNCPAINT means that windows are updated synchronously when invalidated, which generates a WM_PAINT message. Thus, invalidating the window in response to a WM_PAINT message would cause another invalidate, and another WM_PAINT, and so on.

This call requires the existence of a message queue.

WinInvalidateRegion — Invalidate Region

SAA

This call adds a region to a window's update region.

```
WinInvalidateRegion (hwnd, hrgn, includeClippedChildren, Success)
```

Parameters

hwnd *(HWND)* — input
Handle of window whose update region is to be changed:

HWND_DESKTOP	This function applies to the whole screen (or desktop).
Other	Handle of window whose update region is to be changed.

hrgn *(HRGN)* — input
Handle of the region to be added to the window's update region:

NULL	The whole window is to be added into the window's update region.
Other	Handle of the region to be added to the window's update region.

includeClippedChildren *(BOOL)* — input
Invalidation-scope indicator:

TRUE	Include the descendants of **hwnd** in the invalid rectangle
FALSE	Include the descendants of **hwnd** in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_HRGN_BUSY

Remarks

The update region is a subregion of a window that is deemed "invalid" or incorrect in visual terms and is in need of redrawing.

If the window has a CS_SYNCPAINT style, it is redrawn during the processing of this call and the update region should be NULL on return from this call.

If the window has a WS_CLIPCHILDREN style with part of its update region overlapping child windows with a CS_SYNCPAINT style, those children are updated before this call returns.

This function should not be called in response to a WM_PAINT request for windows of style CS_SYNCPAINT. CS_SYNCPAINT means that windows are updated synchronously when invalidated, which generates a WM_PAINT message. Thus, invalidating the window in response to a WM_PAINT message would cause another invalidate, and another WM_PAINT, and so on.

This call requires the existence of a message queue.

WinInvertRect — Invert Rectangle

This call inverts a rectangular area.

WinInvertRect (hps, Rect, *Success*)

Parameters

hps *(HPS)* — input
Presentation-space handle.

The presentation space contains the rectangle to be inverted.

Rect *(RECT)* — input
Rectangle to be inverted.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

Inversion is a logical-NOT operation and has the effect of flipping the bits of each pel.

WinIsChild – Is Child

SAA

This call tests if one window is a descendant of another window.

WinIsChild (Child, Parent, *Related*)

Parameters

Child *(HWND)* – input
Child-window handle.

Parent *(HWND)* – input
Parent-window handle.

Related *(BOOL)* – return
Related indicator:

TRUE Child window is a descendant of the parent window, or is equal to it.
FALSE Child window is not a descendant of the parent, or is an Object Window (even if **Parent** is specified as the desktop or HWND_DESKTOP), or an error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinIsRectEmpty — Is Rectangle Empty

This call checks whether a rectangle is empty.

WinIsRectEmpty (hab, prc, *Empty*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

prc *(RECT)* — input
Rectangle.

Note: The data type *WRECT* may also be used, if supported by the language.

Empty *(BOOL)* — return
Empty indicator:

TRUE Rectangle is empty, or an error occurred
FALSE Rectangle is not empty.

Remarks

A rectangle has area, if its left edge coordinate is less than its right edge coordinate and its bottom edge coordinate is less than its top edge coordinate. An empty rectangle is one with no area.

WinIsThreadActive — Is Thread Active

This call determines whether the active window belongs to the calling execution thread.

```
WinIsThreadActive (hab, Active)
```

Parameters

hab *(HAB)* — input
Anchor-block handle of calling thread.

Active *(BOOL)* — return
Active-window indicator:

TRUE Active window does belong to calling thread
FALSE Active window does not belong to calling thread.

Remarks

This call requires the existence of a message queue.

WinIsWindow — Is Window

This call determines if a window handle is valid.

WinIsWindow (hab, hwnd, *Valid*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

hwnd *(HWND)* — input
Window handle.

Valid *(BOOL)* — return
Validity indicator:

TRUE Window handle is valid
FALSE Window handle is not valid.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinIsWindowEnabled — Query Window Enabled State

SAA

This call returns the enabled state of a window.

WinIsWindowEnabled (hwnd, *Enabled*)

Parameters

hwnd *(HWND)* — input
Window handle.

Enabled *(BOOL)* — return
Enabled-state indicator:

TRUE Window is enabled
FALSE Window is not enabled.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinIsWindowShowing – Query Window Showing

This call determines whether any of the window **hwnd** is physically visible.

WinIsWindowShowing (hwnd, *Showing***)**

Parameters

hwnd *(HWND)* – input
Window handle.

Showing *(BOOL)* – return
Showing state indicator:

TRUE Some part of the window is displayed on the screen
FALSE No part of the window is displayed on the screen.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is useful for applications that constantly output new information. If value FALSE is returned (that is, no part of the window is physically visible), the application can choose not to redraw, since redrawing is not necessary.

If an application is using WinIsWindowShowing, it must issue the call every time it has new information that needs to be updated. If this is not done, invalid screen content could result. The alternative to this approach for a constantly-updating application that has new information is for it to invalidate its window and redraw within a WinBeginPaint - WinEndPaint sequence.

FALSE is returned if the Presentation Manager screen group is not currently visible.

This call requires the existence of a message queue.

WinIsWindowVisible — Query Window Visibility

SAA

This call returns the visibility state of a window.

```
WinIsWindowVisible (hwnd, Visible)
```

Parameters

hwnd *(HWND)* — input
Window handle.

Visible *(BOOL)* — return
Visibility-state indicator:

TRUE Window and all its parents have the WS_VISIBLE style bit set on
FALSE Window or one of its parents have the WS_VISIBLE style bit set off.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

Because **Visible** only reflects the values of WS_VISIBLE style bits, **Visible** may be set to TRUE even if **hwnd** is totally obscured by other windows.

WinLoadAccelTable — Load Accelerator Table

This call loads an accelerator table.

WinLoadAccelTable (hab, Resource, AccelTable, *Accel*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Resource *(RESID)* — input
Resource identity containing the accelerator table.

Module handle returned by the DosLoadModule or DosGetModHandle call referencing a dynamic-link library containing the resource.

AccelTable *(IDENTITY)* — input
Accelerator-table identifier, within the resource file.

Accel *(HACCEL)* — return
Accelerator-table handle.

Possible returns from WinGetLastError:

PMERR_RESOURCE_NOT_FOUND

Remarks

This call returns a different **Accel** value when called twice in succession with the same parameter values.

The accelerator table is owned by the process from which this call is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

WinLoadDlg — Load Dialog

SAA

This call creates a dialog window from the dialog template **DlgId** in **Resource**.

WinLoadDlg (Parent, Owner, DlgProc, Resource, DlgId, CreateParams, *Dlg*)

Parameters

Parent *(HWND)* — input
Parent-window handle of the created dialog window:

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

Owner *(HWND)* — input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified below.

DlgProc *(WNDPROC)* — input
Dialog procedure for the created dialog window.

Resource *(RESID)* — input
Resource identity containing the dialog template.

NULL	Use the application's .EXE file.
Other	Module handle returned from the DosLoadModule or DosGetModHandle call.

DlgId *(IDENTITY)* — input
Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window.

CreateParams *(CREATEPARAMS)* — input
Application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

Dlg *(HWND)* — return
Dialog-window handle:

NULL	Dialog window not created
Other	Dialog window handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND
PMERR_RESOURCE_NOT_FOUND

Remarks

The dialog window is created as an invisible window unless window style WS_VISIBLE is specified in the first dialog-item definition within the dialog template.

The dialog window owner may be modified, in order to ensure sensible results if it is later processed as a modal dialog using the WinProcessDlg or WinGetDlgMsg calls. A search is made up the parent hierarchy, starting at the window specified by the **Owner** parameter, until a child of the window specified by the **Parent** is found. If such a window exists, it is made the actual owner of the dialog. If no such window exists the actual owner of the dialog is set to NULL.

This call returns immediately after creating the dialog window. A WM_INITDLG message is sent to the dialog procedure before this call returns.

This call should not be used while pointing device capture is set. (See WinSetCapture.)

As each of the controls defined within the template of this dialog window is created during the processing of this call, the dialog procedure may receive various control notifications before this call returns.

A dialog window can be destroyed with the WinDestroyWindow call.

Because windows are created from the template, strings in the template are processed with WinSubstituteStrings. Any resultant WM_SUBSTITUTESTRING messages are sent to the dialog procedure before this call returns.

When the child windows of the dialog are created, the WinSubstituteStrings call is used to allow the child windows to perform text substitutions in their window text. If any of the child window text strings contain the percent (%) substitution character, there is an upper limit of 256 on the length of the text string, after it is returned from the substitution.

Programming Note: In general, it is better to create the dialog window invisible as this allows for optimization. In particular, an experienced user can type ahead, anticipating the processing in the dialog window.

In this instance, there may be no need to display the dialog window at all, as the user may have finished the interaction before the window can be displayed.

This is in fact how the WinProcessDlg call works; it does not display the dialog window while there are still WM_CHAR messages in the input queue, but allows these to be processed first.

This call requires the existence of a message queue.

WinLoadHelpTable – Load Help Table

This call identifies the module handle and identity of the help table to the instance of the help manager.

WinLoadHelpTable (HelpInstance, HelpTable, Module, *Success***)**

Parameters

HelpInstance *(HWND)* – input
Handle of a help-manager instance.

This is the handle returned by the WinCreateHelpInstance call.

HelpTable *(IDENTITY)* – input
Identity of the help table.

Module *(RESID)* – input
Handle of the module which contains the help table and help subtable resources.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

An application specifies or changes the handle of the module which contains the help table or the identity of the help table.

This call corresponds to the HM_LOAD_HELP_TABLE message that identifies a help table's identity and the handle of the module which contains the help table and its associated help subtables.

This call requires the existence of a message queue.

WinLoadLibrary – Load Library

This call makes the library available to the application.

WinLoadLibrary (hab, Libname, *Libhandle*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

Libname *(STRL)* – input
Library name.

Libhandle *(HLIB)* – return
Library handle:

NULL Library not successfully loaded
Other Library handle.

Remarks

This call makes the library **Libname** (containing procedures, or resources, or both of these) available to the application.

This call requires the existence of a message queue.

WinLoadMenu — Load Menu

SAA

This call creates a menu window from the menu template **MenuId** from **Resource**, and returns in **Menu** the window handle for the created window.

```
WinLoadMenu (Owner, Resource, MenuId, Menu)
```

Parameters

Owner *(HWND)* — input
Owner- and parent-window handle:

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

Resource *(RESID)* — input
Resource identifier.

NULL	The resource is in the application's .EXE file.
Other	The module handle returned by the DosLoadModule or DosGetModHandle call.

MenuId *(IDENTITY)* — input
Menu identifier within the resource file.

Menu *(HWND)* — return
Menu-window handle.

Remarks

The menu window is created with its parent and owner set to **Owner**, and with identity FID_MENU. If **Owner** is HWND_OBJECT or a window handle returned from WinQueryObjectWindow, the menu window is created as an object window.

Action bar menus are created as child windows of the frame window and are initially visible. Submenus are initially created as object windows that are owned by the window frame.

This call requires the existence of a message queue.

WinLoadPointer — Load Pointer

This call loads a pointer from a resource file into the system.

WinLoadPointer (DeskTop, Resource, Pointer, *hptr*)

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop window
Other	Desktop-window handle returned by WinQueryDesktopWindow.

Resource *(RESID)* — input
Resource identity containing the pointer definition.

NULL	Use the application's own resources file.
Other	Module handle returned by the DosLoadModule or DosGetModHandle call referencing a dynamic-link library containing the resource.

Pointer *(IDENTITY)* — input
Identifier of the pointer to be loaded.

hptr *(HPOINTER)* — return
Pointer handle:

NULL	Error has occurred
Other	Handle of loaded pointer.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_RESOURCE_NOT_FOUND

Remarks

A new copy of the pointer is created each time this call is called. The pointer created by this call can be destroyed using the WinDestroyPointer call. To get one of the standard system pointers, use the WinQuerySysPointer call.

The pointer is owned by the process from which this call is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system.

This call requires the existence of a message queue.

WinLoadProcedure — Load Procedure

This call loads the window or dialog procedure.

```
WinLoadProcedure (hab, Libhandle, Procname, Wndproc)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Libhandle *(HLIB)* — input
Library handle.

Procname *(STRL)* — input
Procedure name.

Wndproc *(WNDPROC)* — return
Window-procedure identifier:

NULL Procedure not successfully loaded
Other Window-procedure identifier.

Remarks

This call loads the window or dialog procedure **Procname** from the library **Libhandle**.

This call requires the existence of a message queue.

WinLoadString — Load String

This call loads a string from a resource.

WinLoadString (hab, Resource, String, BufferMax, Buffer, *Length*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Resource *(RESID)* — input
Resource identity containing the string.

NULL Use the application's own resources file.
Other Module handle returned by the DosLoadModule or DosGetModHandle call referencing a dynamic-link library containing the resource.

String *(IDENTITY)* — input
String identifier.

BufferMax *(SHORT)* — input
Size of buffer.

Buffer *(STRL)* — output
Buffer that is to receive the string.

Length *(SHORT)* — return
The length of the string returned.

This excludes the terminating null, and has the following values:

0 Error
Other A maximum value of (**BufferMax**—1).

Possible returns from WinGetLastError:

PMERR_RESOURCE_NOT_FOUND

Remarks

This call loads a string resource identified by the **String** and the **Resource** parameters into the **Buffer** parameter, and appends a terminating null character.

RT_STRING resources (string resources) contain up to 16 strings each (see "Resource (.RES) File Specification" on page 26-25). The resource ID is calculated from the idString passed to this call as follows:

```
resource ID = (idString / 16) + 1
```

To save storage on disk and in memory, applications should number their string resources sequentially, starting at some multiple of 16.

WinLockHeap – Lock heap

This call converts a heap handle to a pointer to the heap.

```
WinLockHeap (Heap, Start)
```

Parameters

Heap *(HHEAP)* – input
Handle to a heap.

Returned by previous call to the WinCreateHeap call.

Start *(STORAGE)* – return
Segment containing the passed heap.

Possible returns from WinGetLastError:

PMERR_INVALID_HHEAP

Remarks

When memory is allocated from an application data segment, the short pointers returned are directly usable as offsets relative to DS. When memory is allocated from other segments, the short pointers returned must not be referenced relative to the beginning of the segment containing the heap.

Heaps need not be locked; memory management is organized by OS/2.

WinLockVisRegions — Lock Visible Regions

This call locks or unlocks the visible regions of all the windows on the screen, preventing any of the visible regions from changing.

```
WinLockVisRegions (DeskTop, Lock, Success)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle or HWND_DESKTOP.

Lock *(BOOL)* — input
Indicates whether the visible regions are being locked or unlocked:

TRUE Lock the visible regions
FALSE Unlock the visible regions.

Success *(BOOL)* — return
Success indicator:

TRUE Successful
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is useful to threads that wish to prevent window visible regions from changing while it performs some screen operation, such as copying screen pels into a memory bit map.

Any other thread that tries to alter the visible regions is blocked while the visible regions are locked. While the visible regions are locked, no messages must be sent and no functions called that can send messages.

More than one thread can lock the visible regions concurrently. A lock count is maintained by the system, incremented each time the WinLockWindow call is made with **Lock** set to TRUE and decremented each time the WinLockWindow call is made with **Lock** set to FALSE. The visible regions can only change when the lock count is zero.

A thread is only allowed to issue the WinLockWindow call with **Lock** set to FALSE if the corresponding WinLockWindow call with **Lock** set to TRUE is made by a thread of the same process.

This call requires the existence of a message queue.

WinLockWindow — Lock Window

SAA

This call does nothing. It is provided for compatibility with OS/2 Version 1.1.

```
WinLockWindow (hwnd, Lock, Ret)
```

Parameters

hwnd *(HWND)* — input
Window handle.

Lock *(BOOL)* — input
Lock indicator:

TRUE Lock the specified window
FALSE Unlock the specified window.

Ret *(HWND)* — return
Locked-window handle:

NULL Error occurred
Other **hwnd** is returned if successfully locked.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_WINDOW_LOCK_OVERFLOW
PMERR_WINDOW_LOCK_UNDERFLOW

Remarks

Window locking is not required in OS/2 release 1.2 and above.

This call requires the existence of a message queue.

WinLockWindowUpdate – Lock Window Update

This call disables or enables output to a window and its descendants.

WinLockWindowUpdate (DeskTop, LockUpdate, *Success***)**

Parameters

DeskTop *(HWND)* – input
Desktop handle of the screen containing the window to be locked:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

LockUpdate *(HWND)* – input
Handle of window in which output is to be prevented:

NULL	Enable output in the locked window and its descendants.
Other	Handle of the window in which output is to be prevented. Also output is prevented to the descendants of the window.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful operation
FALSE	Error occurred.

Remarks

While the window is disabled, all areas that would have been drawn by applications are remembered and are updated once the window is enabled.

This call is used by threads that wish to draw on an area of the screen over which they have no control. For example, the user interface sizing and moving calls use this call when drawing the shadow box as a window is sized or moved.

All threads continue to run while the window is disabled, only output is prevented.

If one thread disables the window, other threads using this call are blocked until the first enables the window, although they can still receive messages.

This call does not prevent screen group switches, since these may be necessary to handle hard error situations in other screen groups.

This call requires the existence of a message queue.

WinMakePoints — Make Points

This call converts points to graphics points.

```
WinMakePoints (hab, ppt, count, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

ppt *(WPOINT)* — input/output
Points to be converted.

The data type of these points after conversion is *POINT*.

count *(COUNT2)* — input
Number of points to be converted.

Must be positive.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call converts the array of points from a *WPOINT* data structure into a *POINT* data structure.

This call requires the existence of a message queue.

WinMakeRect – Make Rectangle

This call converts a rectangle to a graphics rectangle.

```
WinMakeRect (hab, prc, Success)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

prc *(WRECT)* – input/output
Rectangle to be converted.

The data type of the rectangle after conversion is *RECT*.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This function converts a rectangle from a *WRECT* data structure into a *RECT* data structure.

This call requires the existence of a message queue.

WinMapDlgPoints — Map Dialog Points

SAA

This call maps points from dialog coordinates to window coordinates, or from window coordinates to dialog coordinates.

WinMapDlgPoints (Dlg, Points, Count, Options, *Success***)**

Parameters

Dlg *(HWND)* — input
Dialog-window handle.

Points *(POINT*Count)* — input/output
Coordinate points to be mapped.

The mapped points are substituted.

Count *(USHORT)* — input
Number of coordinate points.

Options *(BOOL)* — input
Calculation control:

TRUE The points are in dialog coordinates and are to be mapped into window coordinates relative to the window specified by the **Dlg** parameter.
FALSE The points are in window coordinates relative to the window specified by the **Dlg** parameter and are to be mapped into dialog coordinates.

Success *(BOOL)* — return
Coordinates-mapped indicator:

TRUE Coordinates successfully mapped
FALSE Coordinates not successfully mapped.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinMapWindowPoints — Map Window Points

This call maps a set of points from a coordinate space relative to one window into a coordinate space relative to another window.

WinMapWindowPoints (From, To, Points, Count, *Success***)**

Parameters

From *(HWND)* — input
Handle of the window from whose coordinates points are to be mapped:

HWND_DESKTOP	Points are to be mapped from screen coordinates
Other	Points are to be mapped from window coordinates.

To *(HWND)* — input
Handle of the window to whose coordinates points are to be mapped:

HWND_DESKTOP	Points are to be mapped into screen coordinates
Other	Points are to be mapped into window coordinates.

Points *(POINT*Count)* — input/output
Points to be mapped to the new coordinate system.

Count *(SHORT)* — input
Number of points to be mapped.

Points can be a *RECT* structure, in which instance this parameter should be set to two.

Programming Note: This is not supported in all languages.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinMessageBox — Message Box

SAA

This call creates, displays, and operates a message box window.

WinMessageBox (Parent, Owner, Text, Title, Window, Style, *Response*)

Parameters

Parent *(HWND)* — input
Parent-window handle of the created message-box window:

HWND_DESKTOP	The message box is to be main window
Other	Parent-window handle.

Owner *(HWND)* — input
Requested owner-window handle of the created message-box window.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg call.

Text *(STRL)* — input
Message-box window message.

The text of the message to be displayed within the message-box window. If multiple lines are required, carriage-return characters must be inserted into the text at appropriate points.

Title *(STRL)* — input
Message-box window title:

The text for the title should not be longer than 40 characters. If text longer than this is supplied, text centering is still performed, even though the beginning and end of the string are not visible.

NULL	The text 'Error' is to be displayed as the title of the message-box window.
Other	The text to be displayed as the title of the message-box window.

Window *(USHORT)* — input
Message-box window identity.

This value is passed to the HK_HELP hook if the WM_HELP message is received by the message-box window.

Style *(BIT16)* — input
Message-box window style.

These values may be combined using the logical-OR operation but only one value can be taken from each of the following groups:

Button or Button Group

MB_OK	Message-box window is to contain an OK pushbutton.
MB_OKCANCEL	Message-box window is to contain both OK and CANCEL pushbuttons.
MB_CANCEL	Message-box window is to contain a CANCEL pushbutton.
MB_ENTER	Message-box window is to contain an ENTER pushbutton.
MB_ENTERCANCEL	Message-box window is to contain both ENTER and CANCEL pushbuttons.
MB_RETRYCANCEL	Message-box window is to contain both RETRY and CANCEL pushbuttons.
MB_ABORTRETRYIGNORE	Message-box window is to contain ABORT, RETRY, and IGNORE pushbuttons.
MB_YESNO	Message-box window is to contain both YES and NO pushbuttons.
MB_YESNOCANCEL	Message-box window is to contain YES, NO, and CANCEL pushbuttons.

Help button

MB_HELP Message-box window is to contain a HELP pushbutton.

When this is selected a WM_HELP message is sent to the window procedure of the message box.

Color or Icon

MB_NOICON Message-box window is not to contain an icon.

MB_ICONHAND Message-box window is to contain a hand icon.

MB_ICONQUESTION Message-box window is to contain a question mark (?) icon.

MB_ICONEXCLAMATION Message-box window is to contain an exclamation point (!) icon.

MB_ICONASTERISK Message-box window is to contain an asterisk (*) icon.

MB_INFORMATION Message-box window is to contain a black information 'i' in a square box.

MB_QUERY Message-box window is to contain a question mark in a square box.

MB_WARNING Message-box window is to contain a black '!' in a square box.

MB_ERROR Message-box window is to contain a STOP sign on a white background.

Default action

MB_DEFBUTTON1 The first button is the default selection. This is the default case, if none of MB_DEFBUTTON1, MB_DEFBUTTON2 and MB_DEFBUTTON3 is specified.

MB_DEFBUTTON2 The second button is the default selection.

MB_DEFBUTTON3 The third button is the default selection.

Modality Indicator

MB_APPLMODAL Message-box window is to be application modal. This is the default case. Its owner is disabled; therefore, do not specify the owner as the parent if this option is used.

MB_SYSTEMMODAL Message-box window is to be system modal.

Mobility Indicator

MB_MOVEABLE Message-box window is to be movable.

The message-box window is displayed with a title bar and a system menu, which shows only the 'Move', 'Close' and 'Task Manager' choices, which can be selected either by use of the pointing device or by accelerator keys.

If the user selects 'Close', the message-box window is removed and the **Response** is set to MBID_CANCEL, whether or not a cancel button existed within the message-box window.

Response *(USHORT)* — return

User-response value:

MBID_ENTER ENTER pushbutton was selected

MBID_OK OK pushbutton was selected

MBID_CANCEL CANCEL pushbutton was selected

MBID_ABORT ABORT pushbutton was selected

MBID_RETRY RETRY pushbutton was selected

MBID_IGNORE IGNORE pushbutton was selected

MBID_YES YES pushbutton was selected

MBID_NO NO pushbutton was selected

MBID_ERROR Function not successful; an error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG

Remarks

The message box consists of a message and a simple dialog with the user.

This call behaves in a similar way to WinDlgBox, and the remarks concerning modality which are documented under that call, and also under the WinLoadDlg and WinProcessDlg calls, also apply here.

This call should not be used while pointing device capture is set (see WinSetCapture).

If the keyboard is used to cycle from one window to the next, the message box and its parent window are considered to be next to each other in the sequence.

If a message box is created as part of the processing of a dialog window, where the dialog window has not been dismissed, the dialog window should be made the owner of the message box window.

If a system modal message box is created to indicate to the user that the system is running out of memory, the strings passed into this call should not be taken from a resource file, as an attempt to load the resource file may fail because of the lack of memory. However, such a message box can safely use the hand icon as this icon is always memory-resident.

The size of the message box is determined as follows:

- The minimum width of a message box is enough to display 40 characters of average width.
- The minimum height of a message box is enough to display 2 lines.
- The text of a message box is word-wrapped by default. If more than two lines are required to display the text, the height of the message box is increased up to a maximum of two thirds of the screen height. The height of a message box can never exceed this value.
- If necessary, the width of a message box is increased to allow room to display the title.

Text is word-wrapped into the message area using the standard rules for a static control; see page 21-1.

The message box is centered on the screen.

If a message box window has a CANCEL button, the MBID_CANCEL value is returned if either the Escape or Cancel keys are pressed. If the message box window has no CANCEL button, pressing the Escape key has no effect.

This call requires the existence of a message queue.

WinModifyDataStructure — Modify Data Structure

This call modifies the contents of a data structure stored in standard form.

WinModifyDataStructure (hab, Count, Types, InLen, InStruc, Handle, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Count *(USHORT)* — input
Number of elements.

This is the number of elements in the **Types** array. It must be zero or greater.

If zero, the data types specified when the data structure was created are used; the **Types** parameter is ignored. Zero may not be specified if the existing data type of the structure is one which has unknown content (STORAGE, for example); data structures can acquire such data types via WinGetMsg and similar calls.

If greater than zero, the data types contained in the **Types** array are used. If the **Types** array contains a data type that does not have an implied length, **Count** must have a value of one; these data types are: BUFFER, STR, and STRL.

Types *(SHORT****Count)** — input
Data type codes.

These define the layout of the input structure **InStruc**, and may be different from those specified when the structure was created. They may be simple data types, structure data types, pointer data types, control data types, or any valid combination. The data types do *not* replace those specified when the structure was originally created; they are used for the current call only.

The data types must not imply a length for the standard form of the structure which exceeds the size specified by the **MaxLen** parameter when the structure was created using the WinCreateDataStructure call. For those data types that do not have an implied length, the value of **InLen** is used.

Note that not all of the data types that occur in the CPI can be specified on this call.

InLen *(USHORT)* — input
Length of input structure.

This is the length in bytes of the input structure **InStruc**. It must not be less than the length implied by the data types that define the structure. For those data types that do not have an implied length, the value of **InLen** determines how much of **InStruc** is used.

InStruc *(BUFFER)* — input
Input structure.

This is the structure in application form, which will be used to modify the standard form of the structure. The application form of the structure depends on the programming language of the application.

In FORTRAN, this parameter must have a *CHARACTER* data type.

Handle *(HSTRUCT)* — input
Handle of data structure to be modified.

WinModifyDataStructure — Modify Data Structure

SAA

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB
PMERR_INV_HSTRUCT

Remarks

The standard form of the data structure may change in size, but it may not exceed the size specified by the **MaxLen** parameter when the structure was created using the WinCreateDataStructure call; the new definition completely replaces the old. The handle of the structure does not change.

Programming Note: This function is required only in COBOL and FORTRAN.

WinMsgMuxSemWait — Message or Multiple Semaphore Wait

Permits an application to wait for an OS/2 semaphore or a Presentation Manager message.

WinMsgMuxSemWait (IndexNbr, List, Timeout, *rc*)

Parameters

IndexNbr *(USHORT)* — output
Index of the cleared semaphore in the list.

List *(STORAGE)* — input
List of semaphore descriptors.

See the DosMuxSemWait function for the construction of a list of semaphore descriptors.

Timeout *(TIME)* — input
Function time-out value.

See the DosMuxSemWait function for a description of the values of this parameter.

rc *(USHORT)* — return
Return code.

See the DosMuxSemWait function for a description of the return codes.

Remarks

This function is identical to the DosMuxSemWait function, except that in addition to waiting on a specified semaphore, window messages sent by the WinSendMsg function from another thread can be received.

Note that only 15 semaphores can be waited for simultaneously, rather than the 16 of the DosMuxSemWait function.

Since the processing of a window message may take longer than the value specified by the **Timeout** parameter, this function may not return within the time specified by that value.

This call requires the existence of a message queue.

WinMsgSemWait – Message or Semaphore Wait

Permits an application to wait for an OS/2 semaphore or a Presentation Manager message.

```
WinMsgSemWait (hsem, Timeout, rc)
```

Parameters

hsem *(HSEM)* – input
Semaphore handle.

See the DosSemWait function for a description of the creation of the semaphore handles.

Timeout *(TIME)* – input
Function time-out value.

See the DosSemWait function for a description of the values of this parameter.

rc *(USHORT)* – return
Return code.

See the DosSemWait function for a description of the return codes.

Remarks

This function is identical to the DosSemWait function, except that in addition to waiting on a specified semaphore, window messages sent by the WinSendMsg function from another thread can be received.

Since the processing of a window message may take longer than the value specified by the **Timeout** parameter, this function may not return within the time specified by that value.

This call requires the existence of a message queue.

WinMultWindowFromIDs — Get Multiple Windows From Identities

This call finds the handles of child windows that belong to a specified window and have window identities within a specified range.

WinMultWindowFromIDs (Parent, hwnd, First, Last, *Windows***)**

Parameters

Parent *(HWND)* — input
Parent-window handle.

hwnd *(HWND***(***Last-First** + 1))* — output
Window handles.

This array must contain (**Last** — **First** + 1) elements. The handle of a window, whose identity is WID (in the range **First** to **Last**), has a zero based index in the array of (WID — **First**). If there is no window for a window identity within the range, the corresponding element in the array is NULL.

First *(USHORT)* — input
First window identity value in the range (inclusive).

Last *(USHORT)* — input
Last window identity value in the range (inclusive).

Windows *(SHORT)* — return
Number of window handles returned:

0	No window handles returned
Other	Number of window handles returned.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call can be used to enumerate all the items in a dialog group, or to enumerate all the frame controls of a standard window. This call is faster than individual calls to the WinWindowFromID call.

WinNextChar — Move to Next Character

This call moves to the next character in a string.

WinNextChar (hab, Codepage, Country, CurrentChar, *NextChar***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Codepage *(USHORT)* — input
Code page.

Country *(USHORT)* — input
Country code.

CurrentChar *(STRL)* — input
Current character in a null-terminated string.

NextChar *(STRL)* — return
Next character in the null-terminated string:

NULL End of string reached
Other Next character.

Possible returns from WinGetLastError:

PMERR_INVALID_STRING_PARM

Remarks

This call handles DBCS strings.

This call requires the existence of a message queue.

WinOffsetRect — Offset Rectangle

This call offsets a rectangle.

WinOffsetRect (hab, rect, cx, cy, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

rect *(RECT)* — input/output
Rectangle to be offset.

Note: The data type *WRECT* can also be used, if supported by the language.

cx *(SHORT)* — input
x value of offset.

cy *(SHORT)* — input
y value of offset.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call offsets the coordinates of **rect** by adding the value of the **cx** parameter to both the left and right coordinates, and the value of the **cy** to both the top and bottom coordinates.

WinOpenClipbrd – Open Clipboard

This call opens the clipboard.

```
WinOpenClipbrd (hab, Success)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

Success *(BOOL)* – return
Success indicator:

TRUE Clipboard successfully opened
FALSE Error occurred.

Remarks

The process reading the clipboard does not become the owner of the object in it; it must not update or free the object.

This call prevents other threads and processes from examining or changing the clipboard contents.

If another thread or process already has the clipboard open, this call does not return until the clipboard is closed.

Messages can be received from other threads and processes during the processing of this call.

This call requires the existence of a message queue.

WinOpenWindowDC — Open Window Device Context

This call opens a device context for a window.

WinOpenWindowDC (hwnd, *hdc*)

Parameters

hwnd *(HWND)* — input
Window handle.

hdc *(HDC)* — return
Device-context handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

hdc is used to associate a presentation space with the window.

Programming Note: The window device context is automatically closed when its associated window is destroyed. It must not be closed with the DevCloseDC call.

The visible region of the device context is updated automatically as windows are rearranged.

Only one device context can be opened for each window. This function must not be issued more than once for the same window.

This call requires the existence of a message queue.

WinPeekMsg — Peek Message

SAA

This call inspects the thread's message queue and returns to the application with or without a message.

WinPeekMsg (hab, msg, Filter, First, Last, Options, *Result***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

msg *(QMSG)* — output
Message structure.

Filter *(HWND)* — input
Window filter.

First *(USHORT)* — input
First message identity.

Last *(USHORT)* — input
Last message identity.

Options *(BIT16)* — input
Options.

If neither of the following flags is specified, the message is not removed. If both of the following flags are specified, the message is removed:

PM_REMOVE Remove message from queue
PM_NOREMOVE Do not remove message from queue.

Result *(BOOL)* — return
Message-available indicator:

TRUE Message available
FALSE No message available.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG

Remarks

This call is identical to the WinGetMsg call, except that it does not wait for the arrival of a message. The message can be left on the queue, by using **Options**.

For details of **Filter**, **First**, and **Last**, see the WinGetMsg call.

The window handle within **msg** is null if the message is posted to the queue with a **hwnd** that is null.

This call requires the existence of a message queue.

WinPostMsg — Post Message

This call posts a message to the message queue associated with the window defined by **hwnd**.

WinPostMsg (hwnd, Msgid, Param1, Param2, *Result*)

Parameters

hwnd *(HWND)* — input
Window handle:

NULL The message is posted into the queue associated with the current thread, and when the message is received by using the WinGetMsg or WinPeekMsg calls, the **hwnd** parameter of the *QMSG* structure is NULL.
Other Window handle.

Msgid *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input
Parameter 1.

Param2 *(MPARAM)* — input
Parameter 2.

Result *(BOOL)* — return
Message-posted indicator:

TRUE Message successfully posted
FALSE Message could not be posted; for example, because the message queue was full.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The message contains **hwnd**, **Msgid**, **Param1**, **Param2**, and the time and pointer position when this function is called.

WinPostQueueMsg — Post Queue Message

SAA

This call posts a message to a message queue.

```
WinPostQueueMsg (hmq, Msgid, Param1, Param2, Success)
```

Parameters

hmq *(HMQ)* — input
Message-queue handle.

Msgid *(USHORT)* — input
Message identifier.

Param1 *(MPARAM)* — input
Parameter 1.

Param2 *(MPARAM)* — input
Parameter 2.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred, or the queue was full.

Possible returns from WinGetLastError:

PMERR_INVALID_HMQ

Remarks

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

It constructs a *QMSG* structure by setting its **hwnd** parameter to NULL, setting its **msgid**, **param1**, and **param2** parameters from the corresponding parameters of this call, and by deriving its **time** and **point** parameters from the system time and pointer position when this function was called. The *QMSG* structure is then placed on the specified queue.

WinPrevChar — Move to Previous Character

This call moves to the previous character in a string.

WinPrevChar (hab, Codepage, Country, Start, CurrentChar, *PrevChar***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Codepage *(USHORT)* — input
Code page.

Country *(USHORT)* — input
Country code.

Start *(STRL)* — input
Character string that contains **CurrentChar**.

CurrentChar *(STRL)* — input
Current character.

PrevChar *(STRL)* — return
Previous character.

The previous character, or the first character if **CurrentChar** is the first character of **Start**.

Possible returns from WinGetLastError:

PMERR_INVALID_STRING_PARM

Remarks

This call handles DBCS strings.

This call requires the existence of a message queue.

WinProcessDlg — Process Modal Dialog

SAA

This call dispatches messages while a modal dialog window is displayed.

WinProcessDlg (Dlg, *Result*)

Parameters

Dlg *(HWND)* — input
Dialog-window handle.

Result *(USHORT)* — return
Reply value.

Value established by the WinDismissDlg call.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

If the dialog has an owner window, that window is disabled. This means that all user input to the owner, and all of its descendants, is prevented.

This call then dispatches messages from the queue to the appropriate window or dialog procedure until the dialog is dismissed by the WinDismissDlg call. This is usually done by the dialog procedure on receipt of an appropriate message, but also occurs if the dialog procedure passes a WM_COMMAND message to WinDefDlgProc or if a WM_QUIT message is encountered before the dialog window is dismissed. In this latter case, WinProcessDlg itself issues a WinDismissDlg call, and posts the WM_QUIT message back to the queue so that the application main loop terminates in the normal way.

This call shows the window, if it is hidden, when the queue is empty. It is therefore possible for the experienced end user to type ahead and cause the dialog to be dismissed before it becomes visible.

The WinDismissDlg call hides the dialog window without destroying it, and also re-enables any window that was disabled by this call.

This call does not return until a WinDismissDlg call is issued in one of the ways listed above. This is true even if the application main window has not been disabled, for example because the dialog window has no owner. In this case, the dialog will appear to the end user to be modeless; the end user will continue to be able to interact with the application, and possibly create multiple instances of the dialog. In such circumstances IBM Operating System/2 calls the application main window procedure recursively before WinProcessDlg returns.

It is not possible to temporarily disable more than one window using this call; a dialog window can have at most one owner. If an application has more than one main window which should be disabled while the modal dialog is displayed, it can be done by setting appropriate hooks using the WinSetHook call.

If the dialog window is a descendant of its owner, this call will disable input to the dialog itself. However, this situation can only occur by explicitly changing the window hierarchy. Dialog windows are created using the WinLoadDlg or WinCreateDlg calls, which modify the owner window specified on their parameter lists.

This call requires the existence of a message queue.

WinPtInRect – Point In Rectangle

This call queries whether a point lies within a rectangle.

```
WinPtInRect (hab, rect, point, f)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

rect *(RECT)* – input
Rectangle to be queried.

Note: The data type *WRECT* can also be used, if supported by the language.

point *(POINT)* – input
Point to be queried.

f *(BOOL)* – return
Success indicator:

TRUE **point** lies within **rect**
FALSE **point** does not lie within **rect**, or an error occurred.

WinQueryAccelTable — Query Accelerator Table

SAA

This call queries the window or queue accelerator table.

WinQueryAccelTable (hab, Frame, *Accel*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Frame *(HWND)* — input
Frame-window handle:

NULL Return queue accelerator.
Other Return the window accelerator table, by sending the WM_QUERYACCELTABLE message to **Frame**.

Accel *(HACCEL)* — return
Accelerator-table handle:

NULL Error occurred
Other Accelerator-table handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinQueryActiveWindow — Query Active Window

This call returns the active window for HWND_DESKTOP, or other parent window.

WinQueryActiveWindow (Parent, Lock, *Active*)

Parameters

Parent *(HWND)* — input
Parent-window handle for which the active window is required:

HWND_DESKTOP	The desktop-window handle that causes this call to return the top-level frame window.
Other	Specified parent-window handle.

Lock *(BOOL)* — input
Windows lock-state indicator:

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

Active *(HWND)* — return
Active-window handle:

NULL	No window is active
Other	Active-window handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinQueryAnchorBlock — Query Anchor Block

This call returns the anchor block handle of the caller.

```
WinQueryAnchorBlock (hwnd, hab)
```

Parameters

hwnd *(HWND)* — input
Window handle.

hab *(HAB)* — return
Anchor block handle.

NULL Invalid **hwnd** parameter
Other Anchor block handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinQueryAtomLength — Query Atom Length

This call queries the length of an atom represented by the specified atom.

```
WinQueryAtomLength (AtomTbl, atom, retlen)
```

Parameters

AtomTbl *(HATOMTBL)* — input
Atom-table handle.

The handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable call.

atom *(ATOM)* — input
Atom whose associated character-string length is to be returned.

retlen *(USHORT)* — return
String length:

0 The specified atom or the atom table is invalid.
Other The length of the character string associated with the atom **excluding** the null terminating byte. Integer atoms always return a length of six.

Possible returns from WinGetLastError:

PMERR_INVALID_HATOMTBL
PMERR_INVALID_ATOM

Remarks

The purpose of this call is to allow an application to determine the size of buffer to use in the WinQueryAtomName call.

WinQueryAtomName — Query Atom Name

This call returns an atom name associated with an atom.

```
WinQueryAtomName (AtomTbl, atom, Buffer, BufferMax, retlen)
```

Parameters

AtomTbl *(HATOMTBL)* — input
Atom-table handle.

The handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable call.

atom *(ATOM)* — input
Atom; identifies the character string to be retrieved.

Buffer *(STRL)* — output
Buffer to receive the character string.

BufferMax *(USHORT)* — input
Buffer size in bytes.

retlen *(USHORT)* — return
Length of retrieved character string:

0 The specified atom or the atom table is invalid.
Other The number of bytes copied to the buffer **excluding** the terminating zero.

Possible returns from WinGetLastError:

PMERR_INVALID_HATOMTBL
PMERR_INVALID_ATOM
PMERR_INVALID_STRING_PARM

Remarks

For integer atoms, the format of the string is "#ddddd" where "ddddd" are decimal digits in the system code page (an ASCII code page). No leading zeros are generated, and the length can be from 3 through 7 characters.

WinQueryAtomUsage — Query Atom Usage

This call returns the number of times an atom has been used.

WinQueryAtomUsage (AtomTbl, atom, *count*)

Parameters

AtomTbl *(HATOMTBL)* — input
Atom-table handle.

The handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable call.

atom *(ATOM)* — input
Atom whose use count is to be returned.

count *(USHORT)* — return
Use count of the atom:

65 535	Integer atom
0	The specified atom or the atom table is invalid
Other	Use count.

Possible returns from WinGetLastError:

PMERR_INVALID_HATOMTBL
PMERR_INVALID_ATOM

WinQueryBits — Query Bits

SAA

This call queries bits in a 4-byte integer, returning the bits as an array of thirty-two integers.

WinQueryBits (hab, lnt, Data, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

lnt *(LONG)* — input
Integer whose bits are to be queried.

Data *(LONG*32)* — output
Data indicating the status of the bits.

0 Bit is off
1 Bit is on.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB

Remarks

The elements of the **Data** parameter are returned in order, with the first element indicating the status of the most significant bit of the **lnt** parameter, and the last element indicating the status of the least significant bit.

Programming Note: This function is required only in COBOL and FORTRAN.

This call requires the existence of a message queue.

WinQueryBitsUnderMask — Query Bits Under Mask

This call queries bits in a 4-byte integer under the control of a mask; the bits identified by the mask are returned as the bit value.

WinQueryBitsUnderMask (hab, lnt, Mask, Bitval, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

lnt *(LONG)* — input
Integer whose bits are to be queried.

Mask *(LONG)* — input
Mask identifying the bits to be queried.

A 'zero' bit in the mask causes the corresponding bit in the **Bitval** parameter to be set to zero.

A 'one' bit in the mask causes the corresponding bit in the **Bitval** parameter to be set equal to the corresponding bit in the **lnt** parameter.

Bitval *(LONG)* — output
Bit value extracted from the integer.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB

Remarks

Programming Note: This function is required only in COBOL and FORTRAN.

WinQueryCapture — Query Capture

This call returns the handle of the window that has the pointer captured.

```
WinQueryCapture (Desktop, Lock, hwnd)
```

Parameters

Desktop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Lock *(BOOL)* — input
Lock-request indicator.

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

hwnd *(HWND)* — return
Handle of the window with the pointer captured:

NULL	No window has the pointer captured, or an error occurred
Handle	Handle of the window with the pointer captured.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinQueryClassInfo — Query Class Information

This call returns window class information.

WinQueryClassInfo (hab, ClassName, ClassInfo, *Exists*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

ClassName *(STRL)* — input
Class name.

ClassInfo *(CLASSINFO)* — output
Class information structure.

Exists *(BOOL)* — return
Class-exists indicator:

TRUE Class does exist
FALSE Class does not exist.

Possible returns from WinGetLastError:

PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND

Remarks

ClassName is either an application-specified name (as defined by the WinRegisterClass call) or the name of a preregistered WC_* class; see page 11-1. Preregistered class names are of the form '#nnnnn', where 'nnnnn' is up to five digits corresponding to the value of the WC_* class name constant.

This call provides information that is needed to create a subclass of a given class (see WinSubclassWindow).

This call requires the existence of a message queue.

WinQueryClassName — Query Class Name

SAA

This call copies the window class name, as a null-terminated string, into a buffer.

WinQueryClassName (hwnd, Length, Buffer, *RetLen***)**

Parameters

hwnd *(HWND)* — input
Window handle.

If this window is of any of the preregistered WC_* classes (see page 11-1), the class name returned in the **Buffer** parameter is in the form "#nnnnn," where "nnnnn" is a group of up to five digits that corresponds to the value of the WC_* class name constant.

Length *(SHORT)* — input
Length of **Buffer**.

Buffer *(STRL)* — output
Class name.

If the class name is longer than **Length**—1, only the first **Length**—1 characters of class name are copied.

RetLen *(SHORT)* — return
Returned class name length.

This is the length, **excluding** the null-termination character.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_STRING_PARM

WinQueryClipbrdData — Query Clipboard Data

This call obtains a handle to the current clipboard data with a specified format.

WinQueryClipbrdData (hab, fmt, *Data*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

fmt *(USHORT)* — input
Format of the data to be accessed.

Data *(ULONG)* — return
Handle to the clipboard data:

NULL Format does not exist, or an error occurred
Other Handle to the clipboard data.

If the memory model is CFI_SELECTOR the low order *USHORT* of this handle is the selector of the data object.

Remarks

The returned data handle must not be used after the WinCloseClipbrd function is called. For this reason, the application must either copy the data (if required for long-term use) or process the data before the WinCloseClipbrd function is called. The application should neither free the data handle itself, nor leave it locked in any way.

Information about the format of the data in the clipboard can be obtained from WinQueryClipbrdFmtInfo.

This call requires the existence of a message queue.

WinQueryClipbrdFmtInfo – Query Clipboard Format Information

This call determines whether a particular format of data is present in the clipboard, and if so, provides information about that format.

```
WinQueryClipbrdFmtInfo (hab, fmt, FmtInfo, Exists)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

fmt *(USHORT)* – input
Format of the data to be queried.

FmtInfo *(USHORT)* – output
Memory model and usage flags.

These are the usage flags set by the setting application; that is, the CFI_* flags of the **FmtInfo** parameter of the WinSetClipbrdData call.

If the format is CF_BITMAP, CF_DSPBITMAP, CF_METAFILE or CF_DSPMETAFILE, **FmtInfo** is set to CFI_HANDLE. If the format is CF_TEXT or CF_DSPTEXT, **FmtInfo** is set to CFI_SELECTOR. If the format is user-defined, **FmtInfo** is set to the value used in the WinSetClipbrdData call.

Exists *(BOOL)* – return
Format-exists indicator:

TRUE **fmt** exists in the clipboard and **FmtInfo** is set
FALSE **fmt** does not exist in the clipboard and **FmtInfo** is not set.

Possible returns from WinGetLastError:

PMERR_INVALID_FLAG

Remarks

This call does not cause the data to be rendered.

This call requires the existence of a message queue.

WinQueryClipbrdOwner — Query Clipboard Owner

This call obtains any current clipboard owner window.

WinQueryClipbrdOwner (hab, Lock, *ClipbrdOwner*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Lock *(BOOL)* — input
Lock state indicator of clipboard owner window.

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

ClipbrdOwner *(HWND)* — return
Window handle of the current clipboard owner:

NULL If the clipboard is not owned by any window, or if an error occurred.
Other Window handle of the current clipboard owner.

Remarks

This call requires the existence of a message queue.

WinQueryClipbrdViewer — Query Clipboard Viewer

This call obtains any current clipboard viewer window.

WinQueryClipbrdViewer (hab, Lock, *ClipbrdViewer*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Lock *(BOOL)* — input
Lock-state indicator of clipboard viewer window.

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

ClipbrdViewer *(HWND)* — return
Current clipboard viewer window handle:

NULL Clipboard does not have a current viewer window, or an error occurred
Other Current clipboard viewer window handle.

Remarks

This call requires the existence of a message queue.

WinQueryCp — Query Code Page

This call returns the queue code page for the specified message queue.

WinQueryCp (hmq, *CodePage*)

Parameters

hmq *(HMQ)* — input
Message queue.

CodePage *(USHORT)* — return
Code page:

0 Error occurred
Other Queue code page for the specified message queue.

Possible returns from WinGetLastError:

PMERR_INVALID_HMQ

Remarks

This call requires the existence of a message queue.

WinQueryCpList — Query Code Page List

This call queries available code pages.

```
WinQueryCpList (hab, count, Codepage, TotCount)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

count *(USHORT)* — input
Maximum number of code pages returned.

Codepage *(USHORT*count)* — output
Code page list.

An array of **count** elements, that contains a list of code pages available to the program.

For more information about code pages, see Chapter 28, Code Pages.

TotCount *(USHORT)* — return
Total number of code pages available:

0 An error occurred
Other Total number of code pages available.

Possible returns from WinGetLastError:

PMERR_PARAMETER_OUT_OF_RANGE

Remarks

This call requires the existence of a message queue.

WinQueryCursorInfo — Query Cursor Information

This call obtains information about any current cursor.

WinQueryCursorInfo (DeskTop, CursorInfo, *Cursor***)**

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

CursorInfo *(CURSORINFO)* — output
Cursor information.

The values are equivalent to the parameters of the WinCreateCursor call except that **rgf** never includes the CURSOR_SETPOS option.

The size and position of the cursor are returned in window coordinates relative to the window identified by the **hwnd** parameter of the structure.

Cursor *(BOOL)* — return
Current-cursor indicator:

TRUE	Cursor exists
FALSE	Cursor does not exist, **CursorInfo** is not updated by this call.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinQueryDataStructure — Query Data Structure

SAA

This call decomposes a data structure stored in standard form into the form required by the application.

WinQueryDataStructure (hab, Handle, Count, Types, BufLen, OutStruc, OutLen, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Handle *(HSTRUCT)* — input
Handle of data structure to be queried.

Count *(USHORT)* — input
Number of elements.

This is the number of elements in the **Types** array. It must be zero or greater.

If zero, the data types specified when the data structure was created are used; the **Types** parameter is ignored. Zero may not be specified if the existing data type of the structure is one which has unknown content (STORAGE, for example); data structures can acquire such data types via WinGetMsg and similar calls.

If greater than zero, the data types contained in the **Types** array are used. If the **Types** array contains a data type that does not have an implied length, **Count** must have a value of one; these data types are BUFFER, and STR.

Types *(SHORT****Count)** — input
Data type codes.

These define the layout of the structure to be used to interpret the standard form. They may be simple data types, structure data types, pointer data types, control data types, or any valid combination.

The data types must not imply a length for the standard form of the structure that exceeds the maximum size of the standard form, specified when the structure was created; that is, it is not possible to query more information than exists in the structure. For data types that do not have an implied length, the value of **BufLen** is used.

Note that not all of the data types that occur in the CPI can be specified on this call.

BufLen *(USHORT)* — input
Length of buffer.

This is the length in bytes of the buffer **OutStruc** available for the output structure. It must not be less than the length of the application form of the structure implied by the data types which define the structure. The value of **BufLen** is used for data types that do not have an implied length.

OutStruc *(BUFFER)* — output
Output structure.

This is the structure in application form. The layout of this structure depends on the programming language of the application.

In FORTRAN, this parameter must have a *CHARACTER* data type.

OutLen *(USHORT)* — output
Length of output structure.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB
PMERR_INV_HSTRUCT

Remarks

The application may decompose structures created explicitly by itself using the WinCreateDataStructure call, and those whose handles are returned to it by PM (for example, by the WinGetMsg call).

When a structure is decomposed, additional handles may be returned by PM, as components in the structure. The application is responsible for destroying *all* data structures whose handles have been returned to it by PM, as well as those which it has itself created. The WinFreeMsg call should be used for message parameters and the message reply, and the WinDestroyDataStructure call for all other structures (unless indicated otherwise).

Note: This call always returns new handles. This means that a given data structure may have more than one handle. In addition to the original handle returned by the WinCreateDataStructure call issued by the application, there may be one or more aliases returned by PM. The alias identifies the same data structure as the original handle, and either may be used to reference, query, or modify the structure. The alias, however, will not permit the structure to increase in size, even if extra space was allocated when the structure was originally created. Also, the alias has as its default data type the data type that applied when it was created by PM, which may be different from the data type(s) used when the original structure was created.

Using the WinDestroyDataStructure call to destroy the alias will leave the original structure and handle intact; destroying the original handle will cause the alias to become invalid, and program errors may result if it is used on any call other than WinDestroyDataStructure.

Programming Note: This function is required only in COBOL and FORTRAN.

This call requires the existence of a message queue.

WinQueryDefinition — Query Definition

This call obtains a copy of the program-information block held for a program or program group.

WinQueryDefinition (hab, ProgHandle, ProgramInfo, MaxLength, *Length*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

ProgHandle *(HPROGRAM)* — input
Handle of the program whose information is to be returned.

ProgramInfo *(PIBSTRUCT)* — output
Program-information data.

This is in the form of a *PIBSTRUCT* data structure. The environment string and parameter string are placed after this in the buffer.

If **ProgHandle** is a group handle, only the program-type and program-title fields are significant.

MaxLength *(USHORT)* — input
Maximum length in bytes:

0 Return the length required to contain the *PIBSTRUCT* structure in the **Length** parameter.
Other Length, in bytes, of data that can be returned in the **ProgramInfo** data structure. This value must be large enough to contain all of the data. If it is not, the call fails.

Length *(USHORT)* — return
Length of returned data:

0 Error occurred
Other Length of data actually returned in the **ProgramInfo** data structure.

Possible returns from WinGetLastError:

PMERR_NOT_IN_IDX
PMERR_INVALID_PROGRAM_HANDLE
PMERR_NOT_CURRENT_PL_VERSION
PMERR_MEMORY_ALLOCATION_ERR
PMERR_MEMORY_DEALLOCATION_ERR
PMERR_BUFFER_TOO_SMALL

Remarks

This call is normally used by passing a **MaxLength** parameter with a value of 0 in order to get the total amount of space required, then to allocating a buffer of the required length. The buffer can then be passed to a second call of this function.

Programming Note: If a *PIBSTRUCT* is passed to this call it fails, as there is insufficient space to copy the help, environment, and parameter strings.

If the target is a program rather than a program group, the data returned in the **ProgramInfo** parameter is in a format that can be used by the WinAddProgram call.

See also the PrfQueryDefinition call, which should be used in preference to this one.

This call requires the existence of a message queue.

WinQueryDesktopWindow — Query Desktop Window

This call returns the desktop-window handle.

WinQueryDesktopWindow (hab, hdc, *DeskTop*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

hdc *(HDC)* — input
Device-context handle:

NULL Default device (the screen).

DeskTop *(HWND)* — return
Desktop-window handle:

NULL Error occurred
Other Desktop-window handle.

Possible returns from WinGetLastError:

PMERR_INV_HDC

Remarks

Only the screen device supports windowing.

Many of the calls that require a desktop-window handle accept HWND_DESKTOP instead. For example, WinCreateWindow accepts HWND_DESKTOP for the parent-window handle to create a main window that is a child of the desktop window.

This call requires the existence of a message queue.

WinQueryDlgItemShort — Query Dialog Item Short

SAA

This call converts the text of a dialog item into an integer value.

```
WinQueryDlgItemShort (Dlg, Item, Result, Signed, Success)
```

Parameters

Dlg *(HWND)* — input
Parent-window handle.

Item *(IDENTITY)* — input
Identity of the child window whose text is to be converted.

Result *(SHORT)* — output
Integer value resulting from the conversion.

Signed *(BOOL)* — input
Sign indicator:

TRUE Signed text, the text is inspected for a minus sign (—)
FALSE Unsigned text.

Success *(BOOL)* — return
Success indicator:

TRUE Successful conversion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is useful for converting a numerical input field into a binary number for further processing. The text of a dialog item is assumed to be an ASCII string.

This call is valid for any window with children. However, it is typically used for dialog items in a dialog window.

This call requires the existence of a message queue.

WinQueryDlgItemText — Query Dialog Item Text

This call queries a text string in a dialog item.

WinQueryDlgItemText (Dlg, Item, MaxText, Text, *RetLen*)

Parameters

Dlg *(HWND)* — input
Parent-window handle.

Item *(IDENTITY)* — input
Identity of the child window whose text is to be queried.

MaxText *(SHORT)* — input
Length of **Text**.

Text *(STRL)* — output
Output string.

This is the text string that is obtained from the dialog item.

RetLen *(USHORT)* — return
Actual number of characters returned:

NULL Error occurred
Other Actual number of characters returned, not including the null-terminating character. The maximum value is **MaxText**—1.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is valid for any window with children. However, it is typically used for dialog items in a dialog window.

This call requires the existence of a message queue.

WinQueryDlgItemTextLength — Query Dialog Item Text Length

SAA

This call queries the length of the text string in a dialog item.

```
WinQueryDlgItemTextLength (Dlg, Item, RetLen)
```

Parameters

Dlg *(HWND)* — input
Parent-window handle.

Item *(IDENTITY)* — input
Identity of the child window whose text is to be queried.

RetLen *(SHORT)* — return
Length of text:

NULL Error occurred
Other Length of text.

Remarks

This call is valid for any window with children. However, it is typically used for dialog items in a dialog window.

This call requires the existence of a message queue.

WinQueryFocus — Query Focus

This call returns the focus window. It is NULL if there is no focus window.

WinQueryFocus (DeskTop, Lock, *Focus***)**

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Lock *(BOOL)* — input
Window lock state indicator.

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

Focus *(HWND)* — return
Focus-handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinQueryHelpInstance — Query Help Instance

This call enables the application to query the instance of the help manager associated with the application-supplied window handle.

```
WinQueryHelpInstance (App, Help)
```

Parameters

App *(HWND)* — input
Handle of the application window.

Help *(HWND)* — return
Help manager window handle:

NULL No help manager instance is associated with the application window
Other Help manager window handle.

Remarks

The help manager first traces the parent window chain until it is NULL or HWND_DESKTOP. Then the help manager traces the owner chain. If a parent of the owner window exists, the trace begins again with the parent chain.

The window chain will be traced until the help manager finds an instance of the help manager or until both the parent and owner windows are NULL or HWND_DESKTOP.

This call requires the existence of a message queue.

WinQueryMsgPos — Query Message Position

This call returns the pointer position, in screen coordinates, when the last message obtained from the current message queue is posted.

```
WinQueryMsgPos (hab, ptrpos, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

ptrpos *(POINT)* — output
Pointer position in screen coordinates.

Success *(BOOL)* — return
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Remarks

The pointer position is the same as that in the **point** parameter of a *QMSG* structure.

To obtain the current position of the pointer, use the WinQueryPointerPos call.

This call requires the existence of a message queue.

WinQueryMsgTime – Query Message Time

This call returns the message time for the last message retrieved by the WinGetMsg or WinPeekMsg calls from the current message queue.

```
WinQueryMsgTime (hab, Time)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

Time *(ULONG)* – return
Time in milliseconds.

Remarks

The message time is the time when the message is posted, measured in milliseconds, from the time when the system is started. Its value is the same as that in the **time** parameter of a *QMSG* structure.

To calculate time delays between messages, subtract the time of the first message from the time of the second message.

Time values are not always increasing, because the value is the number of milliseconds since the system is started, and the system accumulator for this count can wrap through zero.

This call requires the existence of a message queue.

WinQueryObjectWindow — Query Object Window

This call returns the desktop object window handle.

WinQueryObjectWindow (DeskTop, *Object***)**

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Object *(HWND)* — return
Object-window handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

Any window created as a descendant of **Object** is an object window.

This call requires the existence of a message queue.

WinQueryPointer — Query Pointer

SAA

This call returns the pointer handle for **DeskTop**.

```
WinQueryPointer (DeskTop, Pointer)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Pointer *(HPOINTER)* — return
Pointer handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinQueryPointerInfo — Query Pointer Information

This call returns pointer information.

WinQueryPointerInfo (hptr, PointerInfo, *Success*)

Parameters

hptr *(HPOINTER)* — input
Pointer handle.

PointerInfo *(POINTERINFO)* — output
Pointer-information structure.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HPTR

Remarks

The pointer information structure contains information such as the pointer's bit-map handle and action point coordinates. The values returned for the **xhotspot** and the **yhotspot** parameters are in units relative to the size of the system icon or system pointer.

For example, if the application creates a pointer out of a bit map xWide units wide and positions the x coordinate of the pointer's action point at xHot, then this call will return the value of the **xhotspot** as:

```
xHotspot = (xHot * SystemPointerWidth) / xWide
```

where SystemPointerWidth can be obtained by using the WinQuerySysValue call.

This call requires the existence of a message queue.

WinQueryPointerPos — Query Pointer Position

SAA

This call returns the pointer position.

```
WinQueryPointerPos (DeskTop, Point, Success)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Point *(POINT)* — output
Pointer position in screen coordinates.

Success *(BOOL)* — return
Pointer position returned indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The WinQueryMsgPos is used to get the pointer position of the last message obtained by means of the WinGetMsg or WinPeekMsg calls.

This call requires the existence of a message queue.

WinQueryPresParam — Query Presentation Parameter

This call queries the values of presentation parameters for a window.

WinQueryPresParam (hwnd, AttrType1, AttrType2, AttrTypeFound, AttrValueLen, AttrValue, Options, *RetLen*)

Parameters

hwnd *(HWND)* — input
Window handle.

AttrType1 *(IDENTITY4)* — input
First attribute type identity.

This identifies the first presentation parameter attribute to be queried. It can be zero to reference no presentation parameter attribute.

AttrType2 *(IDENTITY4)* — input
Second attribute type identity.

This identifies the second presentation parameter attribute to be queried. It can be zero to reference no presentation parameter attribute.

AttrTypeFound *(IDENTITY4)* — input/output
Attribute type identity found.

This identifies which of the presentation parameter attributes **AttrType1** and **AttrType2** has been found. This parameter can be passed as NULL (if, for example, only one attribute is being queried).

AttrValueLen *(COUNT4B)* — input
Byte count of the size of the **AttrValue** parameter.

AttrValue *(STORAGE)* — output
Attribute value.

The value of the presentation parameter attribute found.

Options *(BIT16)* — input
Options.

Options controlling the query. Any of these can be ORed together.

QPF_NOINHERIT	For the purposes of this query, presentation parameters are not inherited from the owners of the window specified by **hwnd**. If not specified (default), presentation parameters are inherited.
QPF_ID1COLORINDEX	**AttrType1** refers to a color index presentation parameter attribute, the value of which, if found, is to be converted to RGB before being passed back in **AttrValue**.
QPF_ID2COLORINDEX	**AttrType2** refers to a color index presentation parameter attribute, the value of which, if found, is to be converted to RGB before being passed back in **AttrValue**.
QPF_PURERGBCOLOR	Specifies that either or both of **AttrType1** and **AttrType2** reference RGB color, and that these colors must be pure. This is necessary when specifying text foreground and background colors. This is applied after QPF_ID1COLORINDEX and QPF_ID2COLORINDEX.

WinQueryPresParam – Query Presentation Parameter

RetLen *(COUNT4B)* – return
Length of presentation parameter value passed back.

Zero Presentation parameter not found
Other Length of presentation parameter value passed back in **AttrValue**.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

Two presentation parameter attribute identities can be passed, and both will be searched for along the chain of owners of the window **hwnd** (subject to QPF_NOINHERIT). The first one found satisfies the query. If both **AttrType1** and **AttrType2** are present for the same window, **AttrType1** takes precedence.

If the presentation parameter attribute value is too long to fit in the **AttrValue** buffer provided, it is truncated, and the number of bytes copied is returned in **RetLen**. (See also WinSetPresParam and WinRemovePresParam.)

This call requires the existence of a message queue.

WinQueryProfileData — Query Profile Data

This call returns a string of binary data from the initialization file (see Appendix E, "Initialization File Information").

WinQueryProfileData (hab, AppName, KeyName, Value, Size, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

AppName *(STRL)* — input
Application name.

The name of the application for which initialization data is required.

The name has a maximum length of 1024 bytes including the null-termination character. The name must match exactly with the name stored in the file. There is no case-independent searching.

If this parameter equals NULL, this call enumerates all the application names present in the initialization file and returns the names as a list in **Value**. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, **Size** contains the total length of the list excluding the final NULL character.

KeyName *(STRL)* — input
Key name.

The name of the key for which text data is required.

Its length must be less than 1024 bytes including the null-termination character. The name must match exactly with the name stored in the file. There is no case-independent searching.

If this parameter equals NULL, and if **AppName** is not equal to NULL, this call enumerates all key names associated with the named application and returns the key names, but not their values, as a list in **Value**. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, **Size** contains the total length of the list **excluding** the final NULL character.

Value *(STORAGE)* — output
Value data.

A buffer in which the value corresponding to the key name is returned. The returned data is not null terminated, unless the value data is explicitly null terminated within the initialization file. This call handles binary data.

Size *(USHORT)* — input/output
Size of value data.

This is the size of the buffer specified by the **Value** parameter. If the call is successful, this is overwritten with the number of bytes copied into the buffer.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_NOT_IN_IDX
PMERR_CAN_NOT_CALL_SPOOLER

WinQueryProfileData – Query Profile Data

Remarks

This call returns a string of binary data from the initialization file. The call searches the file for a key matching the name specified by **KeyName**, under the application heading specified by **AppName**.

Enumeration can be performed in exactly the same way as in WinQueryProfileString. The enumeration returns application or key names irrespective of whether the data concerned is written with the WinWriteProfileString or the WinWriteProfileData call.

This call returns data that is written to the file using either the WinWriteProfileString or the WinWriteProfileData call.

If **AppName** is NULL, this call enumerates all application names in the initialization file and fills the location pointed to by **Value** with a list of application names. Each application name in the list is terminated with a null character. The last string in the list is terminated with two null characters. This call returns the length of the list, up to, but not including, the final null.

Note: If the enumeration cannot be performed for any reason, the default character string is **not** copied.

If the enumerated application names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with two bytes of zeros, and the call return value is zero. In this case, **KeyName** is ignored.

If **AppName** is valid and if **KeyName** is NULL, this call enumerates all key names associated with **AppName** by filling the location pointed to by **Value** with a list of key names. Each key name in the list is terminated with a null character. The last string in the list is terminated with two null characters.

Note: If the enumeration can not be performed for any reason, the default character string is **not** copied.

This call returns the length of the list, up to, but not including, the final null. If the enumerated key names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with two bytes of zeros, and the call return value is zero.

This call is similar to PrfQueryProfileData with the **hini** parameter set to HINI_PROFILE. PrfQueryProfileData should be used in preference.

This call requires the existence of a message queue.

WinQueryProfileInt — Query Profile Integer

This call retrieves the value of a specified integer key from the initialization file.

WinQueryProfileInt (hab, AppName, KeyName, Default, *RequiredInteger*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

AppName *(STRL)* — input
Application name.

AppName is a text string that is the name of the application for which initialization data is required.

Its length must be less than 1024 bytes including the null-termination character. The search is case-dependent. Names starting with the characters "PM_" are reserved for system use.

KeyName *(STRL)* — input
Key name.

KeyName is a text string that is the name of the key for which integer data is returned.

Its length must be less than 1024 bytes including the null-termination character. The search is case-dependent.

Default *(SHORT)* — input
Default value.

This value is returned in **RequiredInteger**, if the key defined by **KeyName** cannot be found in the initialization file.

RequiredInteger *(SHORT)* — return
Key value specified in the initialization file.

The value of the key specified by **KeyName** in the initialization file.

If the value corresponding to the key is not an integer, **RequiredInteger** is 0.

If the key-name value is a series of digits followed by non-numeric characters, **RequiredInteger** contains the value of the digits only. For example, "KeyName=102abc" causes the value 102 to appear in **RequiredInteger**.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_BUFFER_TOO_SMALL
PMERR_NOT_IN_IDX
PMERR_CAN_NOT_CALL_SPOOLER

Remarks

The initialization file is searched for a key name matching that specified under the application name heading. When an integer is stored as a text string using the WinWriteProfileString call, for example, "123," the returned value is the number, 123. The call returns **Default** if the application-name/key-name pair cannot be found.

Note: The search is case-dependent.

See also the PrfQueryProfileData call, which should be used in preference to this one.

This call requires the existence of a message queue.

WinQueryProfileSize – Query Profile Size

This call obtains the size in bytes of the value of a specified key-name entry for a specified application in the initialization file (see Appendix E, "Initialization File Information").

WinQueryProfileSize (hab, AppName, KeyName, Value, *RetCode*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

AppName *(STRL)* – input
Application name.

The **AppName** parameter is a text string that is the name of the application for which initialization data is required.

Its length must be less than 1024 bytes including the null-termination character.

If **AppName** is NULL, then **Value** returns the length of the buffer required to hold the enumerated list of application names, as returned by WinQueryProfileString when the **AppName** parameter is NULL. In this case, **KeyName** is ignored.

KeyName *(STRL)* – input
Key name.

The **KeyName** parameter is a text string that is the name of the key for which the size of the data is returned.

Its length must be less than 1024 bytes including the null-termination character.

If **KeyName** is NULL, but **AppName** is not NULL, **Value** returns the length of the buffer required to hold the enumerated list of key names for that application name, as returned by WinQueryProfileString when the **KeyName** parameter is NULL, and the **AppName** parameter is not NULL.

Value *(USHORT)* – output
Data length.

The **Value** parameter is the length of the data in the value field related to **KeyName**. If an error occurs, this parameter is undefined.

RetCode *(USHORT)* – return
Success indicator:

0 Successful completion
Other Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_NOT_IN_IDX
PMERR_CAN_NOT_CALL_SPOOLER

Remarks

The **AppName** and **KeyName** parameters must match the names stored in the file exactly. There is no case-independent searching.

This call can be used before using the WinQueryProfileString call, to allocate space for the returned data.

No distinction is made between data that is written using the WinWriteProfileData and the WinWriteProfileString calls.

This call is similar to PrfQueryProfileSize with the **hini** parameter set to HINI_PROFILE. PrfQueryProfileSize should be used in preference.

This call requires the existence of a message queue.

WinQueryProfileString — Query Profile String

This call retrieves a text string from an initialization file.

WinQueryProfileString (hab, AppName, KeyName, Default, ProfileString, MaxPstring, *Pstring*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

AppName *(STRL)* — input
Application name.

A text string that is the name of the application for which initialization data is required.

Its length must be less than 1024 bytes, including the null termination character.

If this parameter equals NULL, this call enumerates all the application names present in the initialization file and returns the names as a list in the **ProfileString** parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the **ProfileString** parameter contains the total length of the list **excluding** the final NULL character.

The search on application name is always case-dependent. Names starting with the characters "PM_" are reserved for system use.

KeyName *(STRL)* — input
Key name.

A text string that is the name of the key for which text data is returned.

Its length must be less than 1024 bytes, including the null termination character.

If this parameter equals NULL, and if the **AppName** parameter is not equal to NULL, this call enumerates all key names associated with the named application and returns the key names (not their values) as a list in the **ProfileString** parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the **ProfileString** parameter contains the total length of the list **excluding** the final NULL character.

The search on key name is always case-dependent.

Default *(STRL)* — input
Default string.

A text string that is returned in the **ProfileString** parameter, if the key defined by the **KeyName** parameter cannot be found in the initialization file.

ProfileString *(STORAGE)* — output
Profile string.

The text string obtained from the initialization file for the key defined by the **KeyName** parameter.

MaxPstring *(USHORT)* — input
Maximum number.

This is the maximum number of characters that can be put into the **ProfileString** parameter, in bytes. If the data from the initialization file is longer than this, it is truncated.

Pstring *(USHORT)* — return
Actual number.

The actual number of characters (including the null termination character) returned in the **ProfileString** parameter, in bytes.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_BUFFER_TOO_SMALL
PMERR_NOT_IN_IDX
PMERR_CAN_NOT_CALL_SPOOLER
PMERR_INVALID_ASCIIZ

Remarks

This call copies a character string from an initialization file, into the **ProfileString** parameter.

The call searches the initialization file for a key matching the name specified by the **KeyName** parameter under the application heading specified by the **AppName** parameter. If the key is found, the corresponding string is copied. If the key does not exist, the default character string, specified by the **Default** parameter, is copied.

If **AppName** is NULL, this call enumerates all the application names in the initialization file and copies into the **ProfileString** parameter a list of application names. Each application name in the list is terminated by a null character. The last string in the list is terminated by two null characters. This call returns the length of the list up to, but not including, the final null.

Programming Note: If the enumeration cannot be performed for any reason, the default character string is **not** copied.

If the enumerated application names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with two bytes of zeros, and the call return value is zero. In this instance, **KeyName** is ignored.

If **AppName** is valid and if **KeyName** is NULL, this call enumerates all key names associated with the **AppName** parameter by copying into the **ProfileString** parameter a list of key names. Each key name in the list is terminated with a null character. The last string in the list is terminated with two null characters.

Programming Note: If the enumeration cannot be performed for any reason, the default character string is **not** copied.

This call returns the length of the list, up to, but not including, the final null. If the enumerated key names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with two bytes of zeros, and the call return value is zero.

This call is case-dependent; thus the strings in the **AppName** parameter and the **KeyName** parameter must match exactly. This avoids any code-page dependency and it is up to the application storing the data to do any case-independent matching.

Programming Note: The application should use WinUpper on the name string before passing it to this function.

The enumeration call does not distinguish between data written with the WinWriteProfileString and the WinWriteProfileData calls.

This call is similar to PrfQueryProfileString with the **hini** parameter set to HINI_PROFILE. PrfQueryProfileString should be used in preference.

This call requires the existence of a message queue.

WinQueryProgramTitles — Query Program Titles

This call obtains information about a program or a program group defined in the program list.

WinQueryProgramTitles (hab, Group, Buffer, BufferLen, Total, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Group *(HPROGRAM)* — input
Handle of the program group whose information is to be returned.

The first group returned is the default group. The handle can also be a program handle rather than a group handle, when a single record of information is returned, that is the data for that program alone.

SGH_ROOT Return the information for the root group.

NULL Return the information for the default group only.

The default group is the one to which a program is added by the WinAddProgram call with its **GroupHandle** parameter set to NULL.

Other Handle of the program group whose information is to be returned.

Buffer *(PROGRAMENTRY****Total)** — output
Program information buffer.

An area of storage into which the program information is returned. This is an array of *PROGRAMENTRY* data structures.

BufferLen *(LENGTH2)* — input
Buffer length.

The total length of **Buffer**, in bytes.

0 Return the total number of *PROGRAMENTRY* data structures in **Total**.
Other Buffer length; however, values between the values of 0 and less than the size of a *PROGRAMENTRY* are invalid.

Total *(USHORT)* — output
Total number of structures.

If **BufferLen** is specified as 0, this is the total number of *PROGRAMENTRY* data structures that are in the program group. Otherwise, this is the total number of *PROGRAMENTRY* data structures actually returned.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_NOT_IN_IDX
PMERR_INVALID_GROUP_HANDLE
PMERR_NOT_CURRENT_PL_VERSION
PMERR_INVALID_TARGET_HANDLE
PMERR_BUFFER_TOO_SMALL

Remarks

Information about all programs in a group is returned in a single operation as an array of entries, one for each program in the group.

This call requires the existence of a message queue.

WinQueryQueueInfo – Query Queue Information

This call returns the information for the specified queue.

```
WinQueryQueueInfo (hmq, Mqinfo, Copied, Success)
```

Parameters

hmq *(HMQ)* – input
Queue handle.

Mqinfo *(MQINFO)* – output
Message queue information structure to contain the queue information.

Copied *(COUNT2B)* – input
Size of message queue information structure that is provided (in bytes).

Specifies the maximum number of bytes to be copied into the **Mqinfo** parameter. This should be the size of an *MQINFO* structure.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call requires the existence of a message queue.

WinQueryQueueStatus — Query Queue Status

This call returns a code indicating the status of the message queue associated with the caller.

WinQueryQueueStatus (DeskTop, *Status*)

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Desktop-window handle returned by WinQueryDesktopWindow.

Status — return
Status information.

Summary *(BIT16)*
Summary of message types existing on the queue.

This field contains a combination of the following values:

QS_KEY	An input event (keyboard or journaling) has caused a WM_CHAR message to be placed in the queue.
QS_MOUSE	An input event has caused a WM_MOUSEMOVE, WM_BUTTON1UP, WM_BUTTON1DOWN, WM_BUTTON1DBLCLK, WM_BUTTON2UP, WM_BUTTON2DOWN, or WM_BUTTON2DBLCLK message to be placed in the queue.
QS_MOUSEBUTTON	An input event has caused a WM_BUTTON1UP, WM_BUTTON1DOWN, WM_BUTTON1DBLCLK, WM_BUTTON2UP, WM_BUTTON2DOWN, or WM_BUTTON2DBLCLK message to be placed in the queue.
QS_MOUSEMOVE	An input event has caused a WM_MOUSEMOVE message to be placed in the queue.
QS_TIMER	A timer event has caused a WM_TIMER message to be placed in the queue.
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 message has been posted to the queue. Note that this message is probably not one of the messages listed above, but could be a WM_CHAR, WM_MOUSEMOVE and so on, message if an application has posted one of these messages. In this case, the corresponding input status flag (QS_KEY, QS_MOUSE, and so on) is not set.
QS_SENDMSG	A message has been sent by another application to a window associated with the current queue.

Added *(BIT16)*
Message type additions.

Message types added to the queue since the last use of this call. The value of this field is a subset of the **Summary** field.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinQueryQueueStatus — Query Queue Status

Remarks

This call is an efficient method for determining whether input is available for processing by the WinGetMsg or WinPeekMsg calls.

This call requires the existence of a message queue.

WinQuerySessionTitle — Query Session Title

This call obtains the title under which a specified application is started, or added to the task list.

WinQuerySessionTitle (hab, Session, Title, Titlelen, *RetCode*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Session *(USHORT)* — input
IBM Operating System/2 session identity of application whose title is requested:

0 Use the session identity of the caller
Other Use the specified session identity.

Title *(STRL)* — output
Task title.

This is the title of the application with a process identity, if the application is present in the task list.

Titlelen *(USHORT)* — input
Maximum length of data returnable in bytes.

If the **Title** parameter is longer than this value it is truncated; however, the terminating null character is left at the end of the string. The maximum number of title characters copied is (**Titlelen**—1).

RetCode *(USHORT)* — return
Return code:

0 Successful completion
Other Error occurred.

Remarks

This call is useful when an application uses the same name in its window title (and in its entry in the task list), as the end-user invokes to start the application. This provides a visual link for the user.

If this call is used after a task list entry is created for the application, the title in the task list entry is obtained. (See also WinQueryTaskTitle.)

WinQuerySwitchEntry — Query Switch Entry

This call obtains a copy of the task list data for a specific application.

```
WinQuerySwitchEntry (Switch, SwitchData, RetCode)
```

Parameters

Switch *(HSWITCH)* — input
Handle to the task list entry.

This can be obtained using the WinQuerySwitchHandle call.

SwitchData *(SWCNTRL)* — output
Switch control data.

Contains information about the specified task list entry. The **hprog** field contains the program handle used to start the program.

RetCode *(USHORT)* — return
Return code.

0 Successful completion
Other Error occurred.

Remarks

This call is available to Presentation Manager and non-Presentation Manager applications.

This call requires the existence of a message queue.

WinQuerySwitchHandle — Query Switch Handle

This call obtains the task list handle belonging to a window.

WinQuerySwitchHandle (hwnd, Process, *Switch*)

Parameters

hwnd *(HWND)* — input
Window handle of an application.

Window handle of an application running in the OS/2 Version 1.2 screen group for which the task list handle is required.

NULL Application is not a OS/2 Version 1.2 application
Other Window handle of an application.

Process *(PID)* — input
Process identity of the application.

Switch *(HSWITCH)* — return
Task list handle for the specified application:

NULL Application is not in the task list, or an error occurred
Other Task list handle.

Remarks

If both a window handle and a process identity are supplied, they must be consistent. Once the task list handle is obtained, it may be used in various other calls to manipulate the task list entry or the program which it references.

This call requires the existence of a message queue.

WinQuerySwitchList — Query Switch List

This call obtains information about the entries in the task list.

```
WinQuerySwitchList (hab, SwitchEntries, DataLength, Count)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

SwitchEntries *(SWBLOCK)* — output
Switch entries block.

Contains a description of all the entries in the current task list. This is held in a *SWBLOCK* structure, which has a count of the number of task list entries, plus a record for each entry containing data such as the process and session identities, the icon handle and the window handle for the running program.

NULL No information returned, however the **Count** parameter contains the total number of task list entries.
Other Task entries block.

DataLength *(USHORT)* — input
Maximum length of data returnable in bytes.

This is the maximum length in bytes of the data that can be returned in the **SwitchEntries** parameter.

0 No information returned, however the **Count** parameter contains the total number of task list entries.
Other Maximum length of data returnable.

Count *(USHORT)* — return
Total number of task list entries present in the system.

NULL Error occurred
Other Total number of task list entries present in the system.

Remarks

It is possible to obtain information about all the currently executing programs in a single operation, with one array entry for each program.

This call requires the existence of a message queue.

WinQuerySysColor — Query System Color

This call returns the system color.

WinQuerySysColor (DeskTop, Color, Reserved, *RgbColor***)**

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Color *(LONG)* — input
System color-index value.

Must be one of the SYSCLR_* index values defined under the WinSetSysColors call.

Reserved *(LONG)* — input
Reserved.

NULL Reserved value; must be zero.

RgbColor *(LONG)* — return
RGB value.

RGB value corresponding to the **Color** parameter.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

This call returns the color value that corresponds to the specified color index of the specified color palette.

WinQuerySysModalWindow — Query System Modal Window

SAA

This call returns the current system modal window.

```
WinQuerySysModalWindow (DeskTop, Lock, SysModal)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Lock *(BOOL)* — input
Window's lock-state indicator.

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

SysModal *(HWND)* — return
Handle of system modal window:

NULL	No system modal window
Other	Handle of system modal window.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

For a full description of the operation of the system modal window, see the WinSetSysModalWindow call.

WinQuerySysPointer — Query System Pointer

This call returns the system-pointer handle.

WinQuerySysPointer (DeskTop, Identifier, Copy, *Pointer*)

Parameters

DeskTop *(HWND)* — input
Desktop-window handle.

Identifier *(SHORT)* — input
System-pointer identifier:

SPTR_ARROW	Arrow pointer
SPTR_TEXT	Text I-beam pointer
SPTR_WAIT	Hourglass pointer
SPTR_SIZE	Size pointer
SPTR_MOVE	Move pointer
SPTR_SIZENWSE	Downward-sloping, double-headed arrow pointer
SPTR_SIZENESW	Upward-sloping, double-headed arrow pointer
SPTR_SIZEWE	Horizontal double-headed arrow pointer
SPTR_SIZENS	Vertical double-headed arrow pointer
SPTR_APPICON	Standard application icon pointer
SPTR_HANDICON	Hand icon pointer
SPTR_QUESICON	Question mark icon pointer
SPTR_BANGICON	Exclamation mark icon pointer
SPTR_NOTEICON	Note icon pointer.

Copy *(BOOL)* — input
Copy indicator:

TRUE	Create a copy of the system pointer and return its handle. Specify this value if the system pointer is to be modified.
FALSE	Return the handle of the system pointer.

Pointer *(HPOINTER)* — return
Pointer handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

Take care when using the pointer bit-map handles returned by the WinQueryPointerInfo call in the *POINTERINFO* structure. If the handle is a system-pointer handle, or is returned by the WinQueryPointerInfo call, it is possible that another application is also accessing the bit-map handle. If this is so, selecting the bit map into a presentation space may fail. Only the active thread may use the bit-map handle returned by either the WinQuerySysPointer call, when **Copy** is FALSE, or by the WinQueryPointerInfo call.

Programming Note: This rule is not enforced by the system; therefore, ensure that the program handles selection failures correctly.

WinQuerySysValue — Query System Value

SAA

This call returns the system value.

```
WinQuerySysValue (DeskTop, ValueId, Value)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	Return the system values for the desktop-window handle
Other	Return the system values for the specified desktop-window handle.

ValueId *(SHORT)* — input
System-value identity.

Dimension values are in pels, and time values are in milliseconds.

Programming Note: Not all system values can be set with WinSetSysValue; those that are changeable are marked with an asterisk (*).

SV_CXSCREEN	Width of screen.
SV_CYSCREEN	Height of screen.
SV_CXVSCROLL	Vertical scroll-bar width.
SV_CYHSCROLL	Horizontal scroll-bar height.
SV_CYVSCROLLARROW	Height of vertical scroll-bar arrow bit maps.
SV_CXHSCROLLARROW	Width of horizontal scroll-bar arrow bit maps.
SV_CYTITLEBAR	Height of caption.
SV_CXBORDER	Width of nominal-width border.
SV_CYBORDER	Height of nominal-width border.
SV_CXSIZEBORDER	(*) Width of sizing border.
SV_CYSIZEBORDER	(*) Height of sizing border.
SV_CXDLGFRAME	Width of dialog-frame border.
SV_CYDLGFRAME	Height of dialog-frame border.
SV_CYVSLIDER	Height of vertical scroll-bar thumb.
SV_CXHSLIDER	Width of horizontal scroll-bar thumb.
SV_CXMINMAXBUTTON	Width of minimize/maximize buttons.
SV_CYMINMAXBUTTON	Height of minimize/maximize buttons.
SV_CYMENU	Single line menu height.
SV_CXFULLSCREEN	Client area width when window is full screen.
SV_CYFULLSCREEN	Client area height when window is full screen (excluding menu height).
SV_CXICON	Icon width.
SV_CYICON	Icon height.
SV_CXPOINTER	Pointer width.
SV_CYPOINTER	Pointer height.

SV_DEBUG	FALSE, indicating not debug system.
SV_CMOUSEBUTTONS	The number of buttons on the pointing device (zero if no pointing device is installed).
SV_POINTERLEVEL	Pointer hide level. If the pointer level is zero, the pointer is visible. If it is greater than zero, the pointer is not visible. The WinShowPointer call is invoked to increment and decrement the SV_POINTERLEVEL, but its value cannot become negative.
SV_CTIMERS	Count of available timers.
SV_SWAPBUTTON	(*) TRUE if pointing device buttons are swapped. Normally, the pointing device buttons are set for right-handed use. Setting this value changes them for left-handed use. If TRUE, WM_LBUTTON* messages are returned when the user presses the right button, and WM_RBUTTON* messages are returned when the left button is pressed. Modifying this value affects the entire system. Applications should not normally read or set this value; users update this value by means of the User Interface Shell to suit their requirements.
SV_CURSORRATE	(*) Cursor blink rate, in milliseconds.
SV_DBLCLKTIME	(*) Pointing device double-click time, in milliseconds.
SV_CXDBLCLK	(*) Width of pointing device double-click sensitive area. The default is the system font character width.
SV_CYDBLCLK	(*) Height of pointing device double-click sensitive area. The default is half the height of the system font character height.
SV_ALARM	(*) TRUE if alarm sound is enabled; FALSE turns off the alarm sound.
SV_WARNINGFREQ	(*) Frequency for warning alarms.
SV_WARNINGDURATION	(*) Duration for warning alarms.
SV_NOTEFREQ	(*) Frequency for note alarms.
SV_NOTEDURATION	(*) Duration for note alarms.
SV_ERRORFREQ	(*) Frequency for error alarms.
SV_ERRORDURATION	(*) Duration for error alarms.
SV_FIRSTSCROLLRATE	(*) The delay in milliseconds, before autoscrolling starts, when using a scroll bar.
SV_SCROLLRATE	(*) The delay in milliseconds, between scroll operations, when using a scroll bar.
SV_CURSORLEVEL	The cursor hide level.
SV_TRACKRECTLEVEL	The hide level of the tracking rectangle (zero if visible, greater than zero if not).
SV_CXBYTEALIGN	Horizontal count of pels for alignment.
SV_CYBYTEALIGN	Vertical count of pels for alignment.
SV_SETLIGHTS	(*) When this is TRUE then the appropriate light will be set when the Keyboard state table is set.

WinQuerySysValue — Query System Value

Value *(LONG)* — return
System value:

0 Error occurred
Other System value.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

ValueId must be one of the SV_* constants; see page 9-224.

WinQuerySystemAtomTable – Query System Atom Table

This call returns the handle of the system atom table.

WinQuerySystemAtomTable (*AtomTbl*)

Parameters

AtomTbl *(HATOMTBL)* – return
System atom-table handle.

Remarks

The system atom table can be accessed by any process in the system. It is created at boot time and cannot be destroyed.

WinQueryTaskSizePos — Query Task Window Size and Position

This call obtains the recommended size, position and status for the first window of a newly-started application (typically the main window).

```
WinQueryTaskSizePos (hab, ScreenGroup, PositionData, RetCode)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

ScreenGroup *(USHORT)* — input
Screen group.

If zero is specified, the screen group number of the caller is used.

PositionData *(SWP)* — output
Window position and size data.

Contains the recommended size and position for the first (main) window of the application. The window flags are set to indicate whether this window should be activated, minimized, or maximized.

RetCode *(USHORT)* — return
Return code:

0	Successful completion
Other	Error occurred.

Remarks

The recommended size, position, and status for the program which is starting up, may be contained in the initialization file. However, if no data is available in the initialization file, the system generates values.

The coordinates returned are screen coordinates.

Note: For a standard window, the values returned apply to the frame window, not the client window. Where generated values are supplied, they are such as to guarantee a non-null client window area within a FS_STANDARD frame window.

This call requires the existence of a message queue.

WinQueryTaskTitle – Query Task Title

This call obtains the title under which a specified application is started, or added to the task list.

WinQueryTaskTitle (Session, Title, Titlelen, *RetCode***)**

Parameters

Session *(USHORT)* – input
Session identity of application whose title is requested:

0 Use the session identity of the caller
Other Use the specified session identity.

Title *(STRL)* – output
Task title.

This is the title of the application with a process identity, if it is present in the task list.

Titlelen *(USHORT)* – input
Maximum length of data returnable in bytes.

If the **Title** parameter is longer than this value it is truncated; however, the terminating null character is left at the end of the string. The maximum number of title characters copied is (**Titlelen**–1).

RetCode *(USHORT)* – return
Return code:

0 Successful completion
Other Error occurred.

Remarks

This call is useful when an application uses the same name in its window title (and in its entry in the task list), as the end-user invokes to start the application. This provides a visual link for the user.

If this call is used after a task list entry is created for the application, the title in the task list entry is obtained. (See also WinQuerySessionTitle.)

This call requires the existence of a message queue.

WinQueryUpdateRect — Query Update Rectangle

SAA

This call returns the rectangle that bounds the update region of a specified window.

WinQueryUpdateRect (hwnd, Prc, *Success*)

Parameters

hwnd *(HWND)* — input
Handle of window whose update rectangle is to be queried.

Prc *(RECT)* — output
Update region that bounds the rectangle (in window coordinates).

Programming Note: The data type *WRECT* can also be used, if supported by the language.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Window has no update region; it is wholly valid, therefore **Prc** is NULL.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is useful for implementing an incremental update scheme as an alternative to the WinBeginPaint and WinEndPaint calls.

This call requires the existence of a message queue.

WinQueryUpdateRegion — Query Update Region

This call obtains an update region of a window.

WinQueryUpdateRegion (hwnd, hrgn, *Complexity*)

Parameters

hwnd *(HWND)* — input
Handle of window whose update region is to be queried.

hrgn *(HRGN)* — input
Handle of the window's update region.

The window's update region, in window coordinates, is copied into **hrgn**.

Complexity *(SHORT)* — return
Complexity of resulting region/error indicator:

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_HRGN_BUSY

Remarks

This call is useful for implementing an alternate update scheme to those used by the WinBeginPaint, and WinEndPaint calls, together with the WinValidateRegion call.

The application can use the returned update region as the clip region for a presentation space, so that drawing output can be clipped to the window's update region.

This call requires the existence of a message queue.

WinQueryValue — Query Value

SAA

This call decomposes a 4-byte integer into 1-byte or 2-byte integers or bit values.

WinQueryValue (hab, Value, Count, Types, Data, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Value *(LONG)* — input
4-byte integer to be decomposed.

Count *(LONG)* — input
Number of elements.

Must be greater than zero, and not more than the number of values that can be extracted from the **Value** parameter, under the control of the **Types** parameter.

Types *(LONG*Count)* — input
Data types of the components of the value to be decomposed.

DTYP_BIT8	8-bit value
DTYP_BIT16	16-bit value
DTYP_BYTE	1-byte unsigned integer
DTYP_UCHAR	1-byte unsigned integer
DTYP_SHORT	2-byte signed integer
DTYP_USHORT	2-byte unsigned integer.

Data *(LONG*Count)* — output
Data components extracted from the value to be decomposed.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB
PMERR_ARRAY_TOO_SMALL
PMERR_ARRAY_TOO_LARGE
PMERR_DATATYPE_ENTRY_INVALID

Remarks

The bytes in the **Value** parameter are processed in order, starting with the most significant byte(s) and ending with the least significant byte(s). During the decomposition of the **Value** parameter, bytes may be skipped to ensure that the components extracted have the correct alignment.

Values whose data types are DTYP_BIT8 or DTYP_BIT16 are returned in the least significant byte(s) in the elements of the **Data** parameter, with the most significant bytes set to null.

Values whose data types are DTYP_BYTE or DTYP_UCHAR or DTYP_USHORT are returned with the most significant bytes in the elements of the **Data** parameter set to null.

Values whose data type is DTYP_SHORT are returned with the sign propagated into the most significant bytes in the elements of the **Data** parameter.

Programming Note: This function is required only in COBOL and FORTRAN.

This call requires the existence of a message queue.

WinQueryVersion — Query Version

SAA

This call returns the version, the revision level and the environment of the Presentation Manager.

```
WinQueryVersion (hab, SysInf)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

SysInf — return
System information within which the application is operating:

SYSINF_ENV *(BIT16)*
Environment:

QV_OS2 OS/2.

SYSINF_MAJVER *(BIT8)*
Major version number of the Presentation Manager.

10 OS/2 Presentation Manager Version 1.

SYSINF_MINVER *(BIT8)*
Minor version (revision) number of the Presentation Manager.

10 Revision 1
20 Revision 2.

WinQueryWindow — Query Window

This call returns the handle of a window that has a specified relationship to a specified window.

WinQueryWindow (hwnd, Code, Lock, *Related*)

Parameters

hwnd *(HWND)* — input
Handle of window to query.

Code *(SHORT)* — input
Type of window information.

Determines what window information is returned:

QW_NEXT Next window in z-order (window below).
QW_PREV Previous window in z-order (window above).
QW_TOP Topmost child window.
QW_BOTTOM Bottommost child window.
QW_OWNER Owner of window.
QW_PARENT Parent of window.
QW_NEXTTOP Returns the next window of the owner window hierarchy subject to their z-ordering.

The enumeration is evaluated in this order:

1. The hierarchy of windows owned by this window in their z-order.
2. The hierarchy of windows of the next z-ordered window having the same owner as this window.
3. The hierarchy of windows in their z-order having the same owner as the owner of this window. This step is repeated until the top of the owner tree for this window is reached.
4. The hierarchy of windows in their z-order of unowned windows.

QW_PREVTOP Returns the previous main window, in the enumeration order defined by QW_NEXTTOP.

Lock *(BOOL)* — input
Lock control.

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

Related *(HWND)* — return
Window handle.

Handle of window related to **hwnd**.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

If this call is used to enumerate windows of other threads, it cannot be ensured that all the windows are enumerated, because the z-ordering of the windows can change during the enumeration. WinGetNextWindow must be used for this purpose.

If this function is called with QW_OWNER or QW_PARENT, the return value is WinQueryDesktopWindow(hab, NULL), and not HWND_DESKTOP, when the desktop window is reached.

If this function is called with QW_PARENT for an object window, the return value is the handle of the object window associated with the desktop window as returned by the WinQueryObjectWindow call.

WinQueryWindowDC – Query Window Device Context

This call returns the device context for a given window.

WinQueryWindowDC (hwnd, *hdc*)

Parameters

hwnd *(HWND)* – input
Window handle.

hdc *(HDC)* – return
Device-context handle:

NULL WinOpenWindowDC has not been called for this window, or an error occurred
Other Device context handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

A handle is only returned if a device context has been opened for the window with WinOpenWindowDC.

This call requires the existence of a message queue.

WinQueryWindowLockCount — Query Window Lock Count

SAA

This call returns the window lock count.

```
WinQueryWindowLockCount (hwnd, Lock)
```

Parameters

hwnd *(HWND)* — input
Window handle.

Lock *(SHORT)* — return
Window-lock count:

0 The window is not locked, or an error occurred
Other Window lock count.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

Window locking is not required in OS/2 Version 1.2 and above.

This call requires the existence of a message queue.

WinQueryWindowPos — Query Window Position

This call queries the window size and position.

WinQueryWindowPos (hwnd, swp, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle.

swp *(SWP)* — output
SWP structure.

The fields are set such that a call to WinSetWindowPos with those values sets the window to its current size and position, with the exception of the **Options** bits which are set as follows:

- SWP_MOVE and SWP_SIZE are set to TRUE.
- SWP_ACTIVATE and SWP_DEACTIVATE, are set to the current state of the window.
- If the window is minimized, SWP_MINIMIZE, is set and SWP_MAXIMIZE, is zero.
- If the window is maximized, SWP_MAXIMIZE, is set and SWP_MINIMIZE, is zero.
- If the window is neither minimized nor maximized, both SWP_MINIMIZE, and SWP_MAXIMIZE, are zero.
- All other bits are set to zero.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG

Remarks

This call requires the existence of a message queue.

WinQueryWindowProcess — Query Window Process

This call obtains the process identity and thread identity of the thread that created a window.

```
WinQueryWindowProcess (hwnd, pid, tid, Success)
```

Parameters

hwnd *(HWND)* — input
Window handle.

pid *(PID)* — output
Process identity of the thread that created the window.

tid *(TID)* — output
Thread identity of the thread that created the window.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

WinQueryWindowPtr — Query Window Pointer

This call retrieves a pointer value from the memory of the reserved window word.

WinQueryWindowPtr (hwnd, b, *p*)

Parameters

hwnd *(HWND)* — input
Window handle.

b *(SHORT)* — input
Index.

Zero-based index of the pointer value to retrieve. The units of **b** are bytes. Valid values are zero through (**Extra** —4), where **Extra** is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage.

The value QWP_PFNWP can be used for the address of the window's window procedure.

p *(STORAGE)* — return
Pointer value.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

This call requires the existence of a message queue.

WinQueryWindowRect — Query Window Rectangle

SAA

This call returns a window rectangle.

WinQueryWindowRect (hwnd, Rect, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle.

Rect *(RECT)* — output
Window rectangle.

Window rectangle of **hwnd**, in window coordinates.

Programming Note: The data type *WRECT* can also be used, if supported by the language.

Success *(BOOL)* — return
Rectangle-returned indicator:

TRUE Rectangle successfully returned
FALSE Rectangle not successfully returned.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The rectangle is in window coordinates relative to itself, so that the bottom-left corner is at the position (0,0) of **hwnd**, or the device if **hwnd** is a main window.

If the size of a frame window has been changed to zero by WinSetWindowPos or WinSetMultWindowPos, the original size is returned because the window is hidden, not sized, in this instance.

WinQueryWindowText — Query Window Text

This call copies window text into a buffer.

WinQueryWindowText (hwnd, Length, Buffer, *RetLen*)

Parameters

hwnd *(HWND)* — input
Window handle.

If **hwnd** is a frame-window handle, the title-bar window text is copied.

Length *(SHORT)* — input
Length.

Length of **Buffer**.

Buffer *(STRL)* — output
Window text.

RetLen *(SHORT)* — return
Length of returned text.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

If the window text is longer than **Length**—1, only the first **Length**—1 characters of window text are copied.

This call sends a WM_QUERYWINDOWPARAMS message to **hwnd**.

If this call references the window of another process, **Buffer** must be in memory that is shared by both processes, otherwise a memory fault can occur.

This call requires the existence of a message queue.

WinQueryWindowTextLength — Query Window Text Length

SAA

This call returns the length of the window text, excluding any null termination character.

WinQueryWindowTextLength (hwnd, *RetLen***)**

Parameters

hwnd *(HWND)* — input
Window handle.

RetLen *(SHORT)* — return
Length of the window text.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This function sends a WM_QUERYWINDOWPARAMS message to **hwnd**.

This call requires the existence of a message queue.

WinQueryWindowULong – Query Window Long

This call obtains the unsigned long integer value, at a specified offset, from the memory of a reserved window word, of a given window.

WinQueryWindowULong (hwnd, b, *Value*)

Parameters

hwnd *(HWND)* – input
Handle of window to be queried.

b *(SHORT)* – input
Index.

Zero-based index into the window words of the value to be queried. The units of **b** are bytes. Valid values are zero through (**Extra** –4), where **Extra** is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. Any of the QWL_* values, are also valid.

Note: QWS_* values cannot be used.

QWL_HMQ	Handle of message queue of window. Note that the leading sixteen bits of this value are zero.
QWL_STYLE	Window style.
QWL_HHEAP	Heap handle used by children of this window.
QWL_HWNDFOCUSSAVE	Window handle of the children of this window that last possessed the focus when this frame window was last deactivated.
QWL_USER	A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes: WC_FRAME (includes dialog windows) WC_LISTBOX WC_BUTTON WC_STATIC WC_ENTRYFIELD WC_SCROLLBAR WC_MENU This value can be used to place application-specific data in controls.
Other	Zero-based index.

Value *(ULONG)* – return
Value contained in the window word.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

The window handle that is passed to this function can be the handle of a window with the same, or different, message queue as the caller, thereby allowing the caller to obtain data from windows belonging to other threads.

WinQueryWindowUShort — Query Window Short

This call obtains the unsigned short integer value at a specified offset from the reserved window word's memory of a given window.

WinQueryWindowUShort (hwnd, b, *Value***)**

Parameters

hwnd *(HWND)* — input
Handle of window to be queried.

b *(SHORT)* — input
Index.

Zero-based index into the window words of the value to be queried. The units of **b** are bytes. Valid values are zero through (**Extra** —2), where **Extra** is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. Any of the QWS_* values, are valid.

Note: QWL_* values cannot be used.

QWS_ID Window identity. The value of the **Id** parameter of the WinCreateWindow call.

QWS_FLAGS These indicators only apply to frame or dialog windows, and contain combinations of the following indicators:

FF_FLASHWINDOW	Frame window is flashing.
FF_ACTIVE	Frame window is displayed in the active state.
FF_SELECTED	Frame window is selected.
FF_FLASHHILITE	Window is currently flashed. This indicator toggles with each flash.
FF_OWNERHIDDEN	Frame window is hidden as a result of its owner being hidden or minimized.
FF_DLGDISMISSED	Dialog has been dismissed by the WinDismissDlg call.
FF_OWNERDISABLED	Window's owner is disabled.

QWS_RESULT Dialog-result parameter, as established by the WinDismissDlg call.

QWS_XRESTORE The x coordinate of the position to which the window is restored.

See also the QWS_CYRESTORE value.

QWS_YRESTORE The y coordinate of the position to which the window is restored.

See also the QWS_CYRESTORE value.

QWS_CXRESTORE The width to which the window is restored.

See also the QWS_CYRESTORE value.

QWS_CYRESTORE The height to which the window is restored.

These values are only valid while the window is maximized or minimized (that is, while either the WS_MINIMIZED or WS_MAXIMIZED window style indicators are set). Changing these values with the WinSetWindowUShort call alters the restore size and position.

QWS_XMINIMIZE The x coordinate of the position to which the window is minimized. If this value is —1, the window has not been minimized.

See also the QWS_YMINIMIZE value.

QWS_YMINIMIZE The y coordinate of the position to which the window is minimized.

When the window is minimized for the first time an arbitrary position is chosen. Changing these values with the WinSetWindowUShort call alters the position of the minimized window, but only when the window is not in a minimized state.

Other Zero-based index.

Value *(USHORT)* — return
Value contained in the indicated window word.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

The window handle that is passed to this call can be the handle of a window with the same, or different, message queue as the caller, thereby allowing the caller to obtain data from windows belonging to other threads.

WinReallocMem — Reallocate Memory In Heap

This call reallocates the size of a memory block on the heap.

WinReallocMem (hHeap, MemObj, Old, New, *Mem*)

Parameters

hHeap *(HHEAP)* — input
Handle to a heap.

This must have been returned from a previous call to WinCreateHeap.

MemObj *(SEGOFF)* — input
Memory object.

The memory block to be reallocated.

Old *(USHORT)* — input
Old size.

Old size of the memory block in bytes.

New *(USHORT)* — input
New size.

New size of the memory block in bytes.

Mem *(SEGOFF)* — return
Pointer to memory block:

Success Short pointer to the reallocated memory.

The short pointer is relative to the beginning of the segment that contains the heap.

The return value can be different from the **MemObj** parameter, if the block is increased in size and cannot be done in place.

NULL Error occurred.

Remarks

The calling routine must specify both the old size of the memory object and the new size. If the new size is larger than the old size, this function calls WinAllocMem to allocate the new, larger object. It also copies **Old** bytes from the old object to the new, frees the old, and returns a pointer to the new object. However, it does not cause an object to grow in place.

The return value of this call is a 16-bit offset of the reallocated memory object, from the start of the segment containing the heap. It returns NULL when it is unable to reallocate the memory object, because an invalid heap handle is specified, there is not enough room in the heap to grow the object to the specified size, or the **MemObj** parameter points to memory outside of the bounds of the passed heap.

The low-order two bits of **Mem** are ignored, even if the memory object is moved as a result of growing, although they are preserved in the return value of this call. Except for the two low bits, the value of the **Mem** parameter must have been returned by either the WinAllocMem call or a previous call to this call.

If the passed heap is created with the HM_MOVEABLE option, the value of the **Old** parameter is ignored, and the value in the size word of the allocated object is used. On completion, the size word contains the value of the **New** parameter. If this call has to move the object in order to satisfy the request, then the handle-value word is updated by adding the amount of the move (in bytes) to it. The returned address is then the address of the first reserved word.

WinRegisterClass — Register Window Class

This call registers a window class.

WinRegisterClass (hab, ClassName, WndProc, ClassStyle, Extra, *Registered*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

ClassName *(STRL)* — input
Window-class name.

An application-specified class name.

WndProc *(WNDPROC)* — input
Window-procedure identifier.

Window-procedure identifier; can be NULL if the application does not provide its own window procedure.

ClassStyle *(BIT32)* — input
Default-window style.

This can be any of the standard class styles (CS_*) (see page 12-1) in addition to any class-specific styles that are defined. These styles can be augmented when a window of this class is created.

A public window class is created if the CS_PUBLIC style is specified, otherwise a private class is created. The CS_PUBLIC style must only be specified on the shell process.

Public classes are available for creating windows from any process. Private classes are only available to the registering process.

Extra *(USHORT)* — input
Reserved storage.

This is the number of bytes of storage reserved per window created of this class for application use.

Registered *(BOOL)* — return
Window-class-registration indicator:

TRUE Window class successfully registered
FALSE Window class not successfully registered.

Possible returns from WinGetLastError:

PMERR_INVALID_FLAG
PMERR_INVALID_INTEGER_ATOM
PMERR_INVALID_HATOMTBL
PMERR_INVALID_ATOM_NAME
PMERR_ATOM_NAME_NOT_FOUND

Remarks

When an application registers a private class with the window procedure in a dynamic link library, it is the application's responsibility to resolve the window-procedure address before issuing this call.

A private class must not be registered with the same name as a public class in the same process.

However, if a private class is registered with the same name as one that already exists, the parameters are ignored but the return value is TRUE. The window procedure of an existing window can be changed using WinSubclassWindow or WinSetWindowPtr. The style of an existing window can be changed with the WinSetWindowULong or WinSetWindowUShort calls. The number of bytes of storage allocated for application use cannot be changed once the window is created.

Private classes are deleted when the process that registers them terminates.

WinRegisterUserDatatype — Register User Data Type

This call registers a data type and defines its structure.

WinRegisterUserDatatype (hab, Datatype, Count, Types, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Datatype *(SHORT)* — input
Data type code to be defined.

This must not be less than DTYP_USER, and must not have been defined previously.

Count *(SHORT)* — input
Number of elements.

Must not be less than one.

Types *(SHORT****Count)** — input
Data type codes of structure components.

Valid data types are the system-defined data types and their pointer equivalents, application-defined data types and their pointer equivalents, and control data types. Note that not all of the data types that occur in the CPI can be specified on this call.

A control data type is followed by one or more entries in the **Types** array which are interpreted in a special way. Control data types allow arrays, offsets, and lengths to be defined.

Simple Data Types:

DTYP_ATOM	See *ATOM* data type.
DTYP_BIT16	See *BIT16* data type.
DTYP_BIT32	See *BIT32* data type.
DTYP_BIT8	See *BIT8* data type.
DTYP_BOOL	See *BOOL* data type.
DTYP_COUNT2	See *COUNT2* data type.
DTYP_COUNT2B	See *COUNT2B* data type.
DTYP_COUNT2CH	See *COUNT2CH* data type.
DTYP_COUNT4B	See *COUNT4B* data type.
DTYP_CPID	See *CPID* data type.
DTYP_ERRORID	See *ERRORID* data type.
DTYP_IDENTITY	See *IDENTITY* data type.
DTYP_IDENTITY4	See *IDENTITY4* data type.
DTYP_INDEX2	See *INDEX2* data type.
DTYP_IPT	See *IPT* data type.
DTYP_LENGTH2	See *LENGTH2* data type.
DTYP_LENGTH4	See *LENGTH4* data type.
DTYP_LONG	See *LONG* data type.
DTYP_OFFSET2B	See *OFFSET2B* data type.
DTYP_PID	See *PID* data type.
DTYP_PIX	See *PIX* data type.
DTYP_PROGCATEGORY	See *PROGCATEGORY* data type.
DTYP_PROPERTY2	See *PROPERTY2* data type.
DTYP_PROPERTY4	See *PROPERTY4* data type.
DTYP_RESID	See *RESID* data type.
DTYP_SEGOFF	See *SEGOFF* data type.
DTYP_SHORT	See *SHORT* data type.
DTYP_TID	See *TID* data type.

WinRegisterUserDatatype — Register User Data Type

DTYP_TIME	See *TIME* data type.
DTYP_UCHAR	See *UCHAR* data type.
DTYP_ULONG	See *ULONG* data type.
DTYP_USHORT	See *USHORT* data type.
DTYP_WIDTH4	See *WIDTH4* data type.
DTYP_WNDPROC	See *WNDPROC* data type.
Handle Data Types:	
DTYP_HAB	See *HAB* data type.
DTYP_HACCEL	See *HACCEL* data type.
DTYP_HAPP	See *HAPP* data type.
DTYP_HATOMTBL	See *HATOMTBL* data type.
DTYP_HBITMAP	See *HBITMAP* data type.
DTYP_HDC	See *HDC* data type.
DTYP_HENUM	See *HENUM* data type.
DTYP_HHEAP	See *HHEAP* data type.
DTYP_HINI	See *HINI* data type.
DTYP_HLIB	See *HLIB* data type.
DTYP_HMF	See *HMF* data type.
DTYP_HMQ	See *HMQ* data type.
DTYP_HPOINTER	See *HPOINTER* data type.
DTYP_HPROGRAM	See *HPROGRAM* data type.
DTYP_HPS	See *HPS* data type.
DTYP_HRGN	See *HRGN* data type.
DTYP_HSEM	See *HSEM* data type.
DTYP_HSPL	See *HSPL* data type.
DTYP_HSWITCH	See *HSWITCH* data type.
DTYP_HVPS	See *HVPS* data type.
DTYP_HWND	See *HWND* data type.
Character/String/Buffer Data Types:	
DTYP_BYTE	See *BYTE* data type.
DTYP_CHAR	See *CHAR* data type.
DTYP_STRL	See *STRL* data type.
DTYP_STR16	See *STR16* data type.
DTYP_STR32	See *STR32* data type.
DTYP_STR64	See *STR64* data type.
DTYP_STR8	See *STR8* data type.
Structure Data Types:	
DTYP_ACCEL	See *ACCEL* data type.
DTYP_ACCELTABLE	See *ACCELTABLE* data type.
DTYP_ARCPARAMS	See *ARCPARAMS* data type.
DTYP_AREABUNDLE	See *AREABUNDLE* data type.
DTYP_BITMAPINFO	See *BITMAPINFO* data type.
DTYP_BITMAPINFOHEADER	See *BITMAPINFOHEADER* data type.
DTYP_BTNCDATA	See *BTNCDATA* data type.
DTYP_CATCHBUF	See *CATCHBUF* data type.
DTYP_CHARBUNDLE	See *CHARBUNDLE* data type.
DTYP_CLASSINFO	See *CLASSINFO* data type.
DTYP_CREATESTRUCT	See *CREATESTRUCT* data type.
DTYP_CURSORINFO	See *CURSORINFO* data type.
DTYP_DEVOPENSTRUC	See *DEVOPENSTRUC* data type.
DTYP_DLGTEMPLATE	See *DLGTEMPLATE* data type.
DTYP_DLGTITEM	See *DLGTITEM* data type.
DTYP_ENTRYFDATA	See *ENTRYFDATA* data type.
DTYP_FATTRS	See *FATTRS* data type.
DTYP_FFDESCS	See *FFDESCS* data type.
DTYP_FIXED	See *FIXED* data type.
DTYP_FONTMETRICS	See *FONTMETRICS* data type.
DTYP_FRAMECDATA	See *FRAMECDATA* data type.
DTYP_GRADIENTL	See *GRADIENTL* data type.
DTYP_HCINFO	See *HCINFO* data type.

DTYP_IMAGEBUNDLE See *IMAGEBUNDLE* data type.
DTYP_KERNINGPAIRS See *KERNINGPAIRS* data type.
DTYP_LINEBUNDLE See *LINEBUNDLE* data type.
DTYP_MARGSTRUCT See *MARGSTRUCT* data type.
DTYP_MARKERBUNDLE See *MARKERBUNDLE* data type.
DTYP_MATRIXLF See *MATRIXLF* data type.
DTYP_MLECTLDATA See *MLECTLDATA* data type.
DTYP_OVERFLOW See *OVERFLOW* data type.
DTYP_OWNERITEM See *OWNERITEM* data type.
DTYP_POINTERINFO See *POINTERINFO* data type.
DTYP_POINTL See *POINTL* data type.
DTYP_PROGRAMENTRY See *PROGRAMENTRY* data type.
DTYP_PROGTYPE See *PROGTYPE* data type.
DTYP_QMSG See *QMSG* data type.
DTYP_RECTL See *RECTL* data type.
DTYP_RGB See *RGB* data type.
DTYP_RGNRECT See *RGNRECT* data type.
DTYP_SBCDATA See *SBCDATA* data type.
DTYP_SIZEF See *SIZEF* data type.
DTYP_SIZEL See *SIZEF* data type.
DTYP_SWBLOCK See *SWBLOCK* data type.
DTYP_SWCNTRL See *SWCNTRL* data type.
DTYP_SWENTRY See *SWENTRY* data type.
DTYP_SWP See *SWP* data type.
DTYP_TRACKINFO See *TRACKINFO* data type.
DTYP_USERBUTTON See *USERBUTTON* data type.
DTYP_WNDPARAMS See *WNDPARAMS* data type.
DTYP_WPOINT See *WPOINT* data type.
DTYP_WRECT See *WRECT* data type.
DTYP_XYWINSIZE See *XYWINSIZE* data type.

Pointer Data Types:

DTYP_Pxxxx Pointer to an item of data type DTYP_xxxx, where DTYP_xxxx is one of the data types in the preceding lists. The value of a pointer data type is the value of the corresponding non-pointer data type prefixed with minus to make it negative.

DTYP_USER Minimum value for application-defined non-pointer data type.

Control Data Types:

DTYP_CTL_ARRAY This starts a sequence of three array elements which define an array; the array resides in the structure being defined, and may have a fixed number of elements, or a variable number of elements.

Element	Value
n	DTYP_CTL_ARRAY
n+1	data type of array data.
n+2	minus the number of elements in the array (for an array of fixed size), or the index of the element in **Types** corresponding to the structure component which contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element must precede element 'n' in **Types**. The index is zero-based.

DTYP_CTL_LENGTH

This starts a sequence of four array elements that define a structure component containing the length of part or all of the structure. The length component resides at this point in the structure.

Element	Value
n	DTYP_CTL_LENGTH
n+1	data type of structure component that contains the length (must be a suitable numeric data type).
n+2	the index of the element in **Types** corresponding to the first structure component that is included in the length; a value of −1 denotes the start of the structure. The index is zero-based. The element specified must not be one which is the second or subsequent element in a DTYP_CTL_* sequence of elements.
n+3	the index of the element in **Types** corresponding to the last structure component that is included in the length; it must not be less than the value contained in element n+2. A value of −1 denotes the end of the structure. The index is zero-based. The element specified must not be one which is the second or subsequent element in a DTYP_CTL_* sequence of elements. If the value is −1, the length includes all offset data residing at the end of the structure.

DTYP_CTL_OFFSET

This starts a sequence of four array elements that define data addressed by an offset. The offset resides at this point in the structure, and contains the offset in bytes of the data from the start of the *outermost* structure in which this component resides. The data addressed by the offset must occupy storage following the fixed part of the structure. The data may be scalar data or array data.

Element	Value
n	DTYP_CTL_OFFSET
n+1	data type of the structure component which contains the offset (must be a suitable unsigned numeric data type).
n+2	data type of offset data.
n+3	minus the number of elements in the array (for an array of fixed size), or the index of the element in **Types** corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. The index is zero-based. A value of −1 indicates that the data is not an array.

DTYP_CTL_PARRAY This starts a sequence of three array elements that define a pointer to an array; the pointer resides at this point in the structure, the array resides elsewhere. The array may have a fixed number of elements, or a variable number of elements.

Element	Value
n	DTYP_CTL_PARRAY
n+1	data type of array data.
n+2	minus the number of elements in the array (for an array of fixed size), or the index of the element in **Types** corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type. The array-size element may occur earlier or later in the structure. The index is zero-based.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB
PMERR_DATATYPE_TOO_SMALL
PMERR_DATATYPE_NOT_UNIQUE
PMERR_ARRAY_TOO_SMALL
PMERR_DATATYPE_ENTRY_INVALID
PMERR_DATATYPE_ENTRY_NOT_NUM
PMERR_DATATYPE_ENTRY_NOT_OFF
PMERR_DATATYPE_ENTRY_BAD_INDEX
PMERR_DATATYPE_ENTRY_CTL_MISS
PMERR_DATATYPE_ENTRY_CTL_BAD

Remarks

This call has no effect unless the RegisterUserMsg hook, which is invoked by this call, has been set.

The value to be used should be obtained by calling WinAddAtom with the handle of the system atom manager, and subtracting DTYP_ATOM_OFFSET from the result.

WinAddAtom is guaranteed to return values in the range 0xC000 to 0xFFFF.

When a data type is defined using this call, a definition for the corresponding pointer data type is automatically established.

This call requires the existence of a message queue.

WinRegisterUserMsg — Register User Message

This call registers a user message and defines its parameters.

WinRegisterUserMsg (hab, Msgid, Type1, Dir1, Type2, Dir2, Typer, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Msgid *(USHORT)* — input
Message identifier.

This must not be less than WM_USER, and must not have been defined previously.

Type1 *(SHORT)* — input
Data type of message parameter 1.

Valid data types are listed below. For data types which are shorter than 4 bytes, the data must be placed in the least-significant bytes, with the most-significant bytes nullified (unsigned data types) or signed-extended (signed data types).

DTYP_BIT16	See *BIT16* data type.
DTYP_BIT32	See *BIT32* data type.
DTYP_BIT8	See *BIT8* data type.
DTYP_BOOL	See *BOOL* data type.
DTYP_LONG	See *LONG* data type.
DTYP_SHORT	See *SHORT* data type.
DTYP_UCHAR	See *UCHAR* data type.
DTYP_ULONG	See *ULONG* data type.
DTYP_USHORT	See *USHORT* data type.
DTYP_P*	A pointer to a system data type. Note that not all of the system data types that exist in the CPI are valid.
< -DTYP_USER	A pointer to a user data type. The user data type must have already been defined via WinRegisterUserDatatype.

Dir1 *(SHORT)* — input
Direction of message parameter 1.

If the message parameter is a pointer, the direction values listed below apply to the *contents* of the storage location pointed at, as well as to the message parameter itself.

RUM_IN	Input parameter (inspected by the recipient of the message, but not altered)
RUM_OUT	Output parameter (altered by the recipient of the message, without inspecting its value first)
RUM_INOUT	Input/output parameter (inspected by the recipient of the message, and then altered).

Type2 *(SHORT)* — input
Data type of message parameter 2.

See the description of **Type1**.

Dir2 *(SHORT)* — input
Direction of message parameter 2.

See the description of **Dir1**.

Typer *(SHORT)* — input
Data type of message reply.

See the description of **Type1**. The message reply is always an output parameter.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB
PMERR_MSGID_TOO_SMALL
PMERR_DATATYPE_INVALID
PMERR_DATATYPE_TOO_LONG

Remarks

This call has no effect unless the RegisterUserMsg hook, which is invoked by this call, has been set.

It is an error to attempt to register the same message identifier more than once within a single OS/2 process.

This call requires the existence of a message queue.

WinRegisterWindowDestroy — Register Window Destroy

This call notifies other applications when this window is destroyed.

WinRegisterWindowDestroy (hwnd, Register, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle.

Register *(BOOL)* — input
Registration indicator:

TRUE Registers the window, so that when it is destroyed, a WM_OTHERWINDOWDESTROYED message is broadcast to all main windows of other tasks. Registering the window is accomplished by incrementing a register count by one.
FALSE Unregisters the window by decrementing the register count by one. The window is not unregistered until the count reaches zero.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinReleaseHook — Release Hook

This call releases an application hook from a hook chain.

WinReleaseHook (hab, hmq, Hook, Address, Module, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

hmq *(HMQ)* — input
Handle of message queue from which the hook is to be released:

HMQ_CURRENT	The hook is released from the message queue associated with the current thread (calling thread).
NULL	The hook is released from the system hook chain.

Hook *(SHORT)* — input
Type of hook chain. This must be one of the HK_* values; for details, see Chapter 10, "Functions Supplied by Applications."

Address *(PROC)* — input
Address of the hook routine.

Module *(RESID)* — input
Module handle:

NULL	The hook procedure is in the application's .EXE file.
Module	This is the module that contains the application procedure, as returned by the DosLoadModule or DosGetModHandle call.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HMQ
PMERR_PARAMETER_OUT_OF_RANGE

WinReleasePS — Release Presentation Space

SAA

This call releases a cache presentation space obtained using the WinGetPS or the WinGetScreenPS call.

WinReleasePS (hps, *Success*)

Parameters

hps *(HPS)* — input
Handle of the cache presentation space to release.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

Only cache presentation spaces can be released using this call, after which the presentation space is returned to the cache to be used again. The presentation-space handle should not be used following this call.

WinRemovePresParam — Remove Presentation Parameter

This call removes a presentation parameter that is associated with the window **hwnd**.

WinRemovePresParam (hwnd, AttrType, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle.

AttrType *(IDENTITY4)* — input
Attribute type identity.

The type of the presentation parameter attribute that is to be removed.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

See also WinSetPresParam and WinQueryPresParam.

This call requires the existence of a message queue.

WinRemoveSwitchEntry — Remove Switch Entry

This call removes a specified entry from the task list.

```
WinRemoveSwitchEntry (Switch, RetCode)
```

Parameters

Switch *(HSWITCH)* — input
Task-list entry handle.

RetCode *(USHORT)* — return
Success indicator:

0 Successful completion
Other Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_SWITCH_HANDLE
PMERR_INVALID_WINDOW

Remarks

An application that uses OS/2 Version 1.2 effectively should, at least, add its main window to the task list when it starts, and remove it from the task list when it stops.

Task-list entries for non-OS/2 Version 1.2 applications cannot be removed using this call. These entries are removed automatically by the system when the screen group they occupy terminates.

Note: This call and the WinCreateSwitchEntry and WinAddSwitchEntry calls are not required if the main window is created with the frame style FCF_TASKLIST or FCF_STANDARD, as these styles automatically update the task list when the main window is created, destroyed, or its title changes.

This call requires the existence of a message queue.

WinScrollWindow — Scroll Window

This call scrolls the contents of a window rectangle.

WinScrollWindow (hwnd, Dx, Dy, Scroll, Clip, UpdateRgn, Update, Options, *Complexity***)**

Parameters

hwnd *(HWND)* — input
Window handle.

Dx *(SHORT)* — input
Amount of horizontal scroll to the right (in device units).

Dy *(SHORT)* — input
Amount of vertical scroll upward (in device units).

Scroll *(RECT)* — input
Scroll rectangle.

If this is NULL, the entire window is scrolled.

Programming Note: The data type *WRECT* can also be used, if supported by the language.

Clip *(RECT)* — input
Clip rectangle.

If not NULL, this defines a clip rectangle that clips the destination of the scroll.

UpdateRgn *(HRGN)* — input
Update region.

If not NULL, this contains the region uncovered by the scroll when returned.

Update *(RECT)* — input/output
Update rectangle.

If not NULL, this contains the bounding rectangle of the invalid bits uncovered by the scroll when returned.

Options *(BIT16)* — input
Scroll options.

SW_SCROLLCHILDREN
Unless this is set, child windows are not scrolled. If this is set, and **Scroll** is NULL, all the child windows are scrolled by **Dx** and **Dy** units. If **Scroll** is not NULL, only those child windows that intersect **Scroll** are scrolled.
SW_INVALIDATERGN
The invalid region created as a result of the scroll is added to the update regions of those windows affected. This may result in sending WM_PAINT messages to CS_SYNCPAINT windows before the call returns.

Complexity *(SHORT)* — return
Complexity of resulting region/error indicator:

RGN_NULL	NULL rectangle invalid
RGN_RECT	Simple rectangle invalid
RGN_COMPLEX	Complex rectangle invalid
RGN_ERROR	Error.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG
PMERR_HRGN_BUSY

WinScrollWindow — Scroll Window

SAA

Remarks

This function scrolls the contents of a rectangle defined by **Scroll** in the window **hwnd**, by **Dx** units horizontally and **Dy** units vertically. All coordinates must be in device units.

Clipping takes place on the final image of the scrolling. Even if the scroll rectangle lies outside the clip rectangle, these bits are scrolled, if their destination lies within the intersection of the clip rectangle and the destination rectangle.

This function returns an RGN_* value, indicating the type of invalid region created by the scroll as returned by GpiCombineRegion. RGN_ERROR is returned if **hwnd** is invalid.

Programming Note: If **hwnd** has style WS_CLIPCHILDREN, portions of any child window within the scroll area are scrolled. In this instance, this function must be called with **Options** SW_SCROLLCHILDREN.

This is the only function that can be used by a thread to move bits within its own window, because of the critical section nature of window update regions.

Scrolling is fastest without SW_SCROLLCHILDREN and SW_INVALIDATERGN. When scrolling needs to be repeated quickly, do not include the SW_INVALIDATERGN flag and repaint the invalid area if the invalid region is rectangular, otherwise invalidate and update.

If the scrolling is infrequent, include the SW_INVALIDATERGN flag. This function invalidates and updates synchronous-paint windows automatically before returning.

The cursor and the track rectangle are scrolled when they intersect with the scrolled region. Any part of the window's initial update region that intersects the scrolled region is also offset.

This call requires the existence of a message queue.

WinSendDlgItemMsg — Send Message to Dialog Item

This call sends a message to the dialog item defined by **Item** in the dialog window specified by **Dlg**.

WinSendDlgItemMsg (Dlg, Item, Msgid, Param1, Param2, *Reply*)

Parameters

Dlg *(HWND)* — input
Parent-window handle.

Item *(IDENTITY)* — input
Identity of the child window.

Msgid *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input
Message parameter 1.

Param2 *(MPARAM)* — input
Message parameter 2.

Reply *(MRESULT)* — return
Message-return data.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is equivalent to the WinSendMsg call, in which the receiving window procedure is specified by means of the item identity of the child window and parent-window handle.

It does not return until the message has been processed by the dialog item, whose return value is returned in **Reply**.

The call is equivalent to:

```
WinSendMsg (WinWindowFromID(dlg, item, hwnd), msgid, param1, param2, reply);
```

This call is valid for any window with children; however, it is typically used for dialog items in a dialog window.

This call requires the existence of a message queue.

WinSendMsg — Send Message

SAA

This call sends a message with identity **Msgid** to **hwnd**, passing **Param1** and **Param2** as the parameters to the window.

WinSendMsg (hwnd, Msgid, Param1, Param2, *Reply*)

Parameters

hwnd *(HWND)* — input
Window handle.

Msgid *(USHORT)* — input
Message identity.

Param1 *(MPARAM)* — input
Parameter 1.

Param2 *(MPARAM)* — input
Parameter 2.

Reply *(MRESULT)* — return
Message-return data.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_WINDOW_NOT_LOCKED

Remarks

Reply is the value returned by the window procedure that is invoked. For standard window classes, the values of **Reply** are documented with the message definitions.

This call does not complete until the message has been processed by the window procedure whose return value is returned in **Reply**.

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, IBM Operating System/2 switches to the appropriate thread that enters the necessary window procedure recursively. The message is not placed in the destination thread's queue.

This call requires the existence of a message queue.

WinSetAccelTable — Set Accelerator Table

This call sets the window-accelerator, or queue-accelerator table.

WinSetAccelTable (hab, Accel, Frame, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Accel *(HACCEL)* — input
Accelerator-table handle:

NULL Remove any accelerator table in effect for the window or the queue
Other Accelerator-table handle.

Frame *(HWND)* — input
Frame-window handle:

NULL Set the queue-accelerator table
Other Set the window-accelerator table.

Success *(BOOL)* — return
Success Indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_HACCEL

Remarks

This call requires the existence of a message queue.

WinSetActiveWindow — Set Active Window

SAA

This call makes the main window the active window.

```
WinSetActiveWindow (DeskTop, hwnd, Success)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hwnd *(HWND)* — input
Window handle.

hwnd does not have to be a main window itself. If it is not, the active window is set to the main window that is the ultimate parent of **hwnd**.

Success *(BOOL)* — return
Active-window-set indicator:

TRUE	Active window is set
FALSE	Active window is not set.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is equivalent to the WinFocusChange call in which the **FocusChange** parameter is set to FC_SETACTIVEFOCUS.

This call requires the existence of a message queue.

WinSetBits — Set Bits

This call modifies bits in a 4-byte integer under the control of an array of thirty-two integers.

WinSetBits (hab, Int, Data, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Int *(LONG)* — input/output
Integer whose bits are to be modified.

Data *(LONG*32)* — input
Data used to modify the integer:

0 Bit is set off
>0 Bit is set on
<0 Bit is not altered.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB

Remarks

The elements of the **Data** parameter are processed in order, so that the first element controls the most significant bit within the **Int** parameter, and the last element controls the least significant bit.

Programming Note: This function is required only in COBOL and FORTRAN.

This call requires the existence of a message queue.

WinSetBitsUnderMask — Set Bits Under Mask

SAA

This call modifies bits in a 4-byte integer under the control of a mask; the bits identified by the mask are set equal to the corresponding bits in the bit value supplied.

```
WinSetBitsUnderMask (hab, int, Mask, Bitval, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

int *(LONG)* — input/output
Integer whose bits are to be modified.

Mask *(LONG)* — input
Mask identifying the bits to be modified.

A 'zero' bit in the mask preserves the existing setting of the corresponding bit in the **int** parameter.

A 'one' bit in the mask causes the corresponding bit in the **int** parameter to be set equal to the corresponding bit in the **Bitval** parameter.

Bitval *(LONG)* — input
Bit value to be used to modify the integer.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB

Remarks

Programming Note: This function is required only in COBOL and FORTRAN.

This call captures all pointing device messages.

WinSetCapture (Desktop, hwnd, *Success***)**

Parameters

Desktop *(HWND)* — input
Desktop-window handle, or HWND_DESKTOP.

hwnd *(HWND)* — input
Handle of the window that is to receive all pointing device messages.

hwnd can take the special value HWND_THREADCAPTURE to capture the pointing device to the current thread rather than to a particular window. HWND_THREADCAPTURE is unique among window handles.

If **hwnd** is NULL, pointing device capture is released.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred. If the pointing device has already been captured by another thread or window, the call fails. This is to prevent applications removing the capture from other windows or threads.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call assigns the pointing device capture to **hwnd**.

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

When this function (**Desktop**, NULL) is called to release the pointing device capture, a WM_MOUSEMOVE message is posted regardless of whether the pointing device pointer has actually moved. This ensures that the window below the pointing device, at that time, is able to change features, such as the shape of the pointing device pointer.

If this function (**Desktop**, HWND_THREADCAPTURE) is called, the pointing device is captured to the current thread. Pointing device QMSGs processed in this manner have NULL window handles, and the pointing device coordinates are relative to the screen.

This function returns an unlocked window handle.

It must only be called while processing pointing device or keyboard input. A message box or dialog box must not be created while the pointing device is captured.

This call requires the existence of a message queue.

WinSetClassMsgInterest — Set Class Message Interest

SAA

This call sets the message interest of a window class.

WinSetClassMsgInterest (hab, ClassName, MsgClass, Control, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

ClassName *(STRL)* — input
Window-class name.

MsgClass *(USHORT)* — input
Message class to have interest level set:

msgId	A single message identity (for example, WM_SHOW)
SMIM_ALL	All messages.

Control *(SHORT)* — input
Interest identifier for the message class:

SMI_INTEREST	Interested in the message, or messages
SMI_NOINTEREST	Not interested in the message, or messages
SMI_AUTODISPATCH	Interested in the message or messages, but they are to be automatically dispatched to the window procedure.

Success *(BOOL)* — return
Interest-changed indicator:

TRUE	Interest successfully changed
FALSE	Interest not successfully changed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call has no effect unless the MsgCtlHook hook, which is invoked by this call, has been set.

This call requires the existence of a message queue.

WinSetClipbrdData — Set Clipboard Data

This call puts data into the clipboard.

WinSetClipbrdData (hab, h, fmt, FmtInfo, *DataPlaced*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

h *(ULONG)* — input
Handle.

General handle to the data object being set into the clipboard. If NULL, a WM_RENDERFMT message is sent to the clipboard-owner window to render the format when WinQueryClipbrdData is called with the specified format.

Once the data has been set into the clipboard, this handle can no longer be used by the application.

If CFI_SELECTOR is specified, this parameter contains the selector in the low-order USHORT. The selector must be shareable, that is, it must have been created as 'shareable through DosGiveSeg' in either the DosAllocSeg or DosAllocHuge call. If DosAllocHuge was used, this parameter should specify the selector returned by this call.

fmt *(USHORT)* — input
Format.

Clipboard format of the data object referenced by **h**.

The standard clipboard formats are shown in the following list. In addition to these predefined formats, any format value registered through the standard system atom manager displays this format in preference to privately-formatted data.

CF_TEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
CF_DSPTEXT	Text display format associated with private format.
CF_BITMAP	Bit map.
CF_DSPBITMAP	Bit-map display format associated with private format.
CF_METAFILE	Metafile.
CF_DSPMETAFILE	Metafile display format associated with private format.

FmtInfo *(BIT16)* — input
Information.

Memory Model Name

CFI_SELECTOR	Handle is a selector in the low order *USHORT* Calls return the data object with segment in the low order USHORT, and ZERO in the high order USHORT. This must be manipulated into a far pointer before use, otherwise a protection fault occurs.
CFI_HANDLE	Handle is the handle to a metafile or bit map.

Usage Flags

CFI_OWNERFREE	Handle is not freed by WinEmptyClipbrd. The application must free the data if necessary.

WinSetClipbrdData – Set Clipboard Data

CFI_OWNERDISPLAY This flag indicates that the format is drawn by the clipboard owner in the clipboard viewer window by means of the WM_PAINTCLIPBOARD message. **h** should be NULL.

Information about the type of data referenced by **h**.

Using This Function for User-Defined Formats

When an application sets a user-defined format into the clipboard, it can specify the CFI_SELECTOR memory model. If it does, the system:

- saves the selector (accessing it from the shell process), so that if the setting application terminates normally or abnormally, the data is still available.
- frees the selector from the setting process, so that the setting application may no longer use it.

If the application does not specify the memory model, no action is taken.

DataPlaced *(BOOL)* – return
Data-placed indicator.

Indicates whether data is placed into clipboard by this call:

TRUE Data placed into clipboard.
FALSE Data is not placed into clipboard, either an error occurred, or **h** is NULL.

Possible returns from WinGetLastError:

PMERR_INVALID_FLAG
PMERR_INVALID_INTEGER_ATOM

Remarks

Data of the specified format, already in the clipboard, is freed by this call.

An object passed to the clipboard becomes the property of the system, and is not deleted when the process that created it terminates.

This call requires the existence of a message queue.

WinSetClipbrdOwner — Set Clipboard Owner

This call sets the current clipboard-owner window.

WinSetClipbrdOwner (hab, hwnd, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

hwnd *(HWND)* — input
Window handle of the new clipboard owner:

NULL Clipboard-owner window is released and no new clipboard-owner window is established.
Other Window handle of the new clipboard owner.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The clipboard owner window receives the following clipboard-related messages at appropriate times:

WM_DESTROYCLIPBOARD
WM_HSCROLLCLIPBOARD
WM_PAINTCLIPBOARD
WM_RENDERFMT
WM_RENDERALLFMTS
WM_SIZECLIPBOARD
WM_VSCROLLCLIPBOARD

This call requires the existence of a message queue.

WinSetClipbrdViewer — Set Clipboard Viewer

This call sets the current clipboard-viewer window to a specified window.

```
WinSetClipbrdViewer (hab, NewClipViewer, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

NewClipViewer *(HWND)* — input
Window handle of the new clipboard viewer:

NULL The clipboard-viewer window is released and no new clipboard-viewer window is established.
Other Window handle of the new clipboard viewer.

Success *(BOOL)* — return
Success indicator:

TRUE Valid, new clipboard-viewer window established
FALSE There is no new clipboard-viewer window established.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The clipboard-viewer window receives the WM_DRAWCLIPBOARD message when the contents of the clipboard change. This allows the viewer window to display an up-to-date version of the clipboard contents.

The clipboard must be open before this call is invoked.

This call requires the existence of a message queue.

WinSetCp – Set Code Page

This call sets the code page for a queue.

WinSetCp (hmq, CodePage, *Success*)

Parameters

hmq *(HMQ)* – input
Message-queue handle.

CodePage *(USHORT)* – input
Code page.

Either of the two ASCII code pages specified in CONFIG.SYS can be selected.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HMQ
PMERR_RESOURCE_NOT_FOUND

Remarks

This call requires the existence of a message queue.

WinSetDlgItemShort — Set Dialog Item Short

SAA

This call converts an integer value into the text of a dialog item.

```
WinSetDlgItemShort (Dlg, Item, Value, Signed, Success)
```

Parameters

Dlg *(HWND)* — input
Parent-window handle.

Item *(IDENTITY)* — input
Identity of the child window whose text is to be changed.

Value *(USHORT)* — input
Integer value used to generate the dialog item text.

Signed *(BOOL)* — input
Sign indicator:

TRUE Signed integer value
FALSE Unsigned integer value.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The text produced is an ASCII string.

This call is valid for any window with children; however, it is typically used for dialog items in a dialog window.

This call requires the existence of a message queue.

WinSetDlgItemText — Set Dialog Item Text

This call sets a text string in a dialog item.

WinSetDlgItemText (Dlg, Item, Text, *Success*)

Parameters

Dlg *(HWND)* — input
Parent-window handle.

Item *(IDENTITY)* — input
Identity of the child window whose text is to be set.

Text *(STRL)* — input
Source string.

This is the text string which is to be set into the dialog item.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is valid for any window with children. However, it is typically used for dialog items in a dialog window.

This call requires the existence of a message queue.

WinSetErrorInfo — Set Error Information

This call sets detailed error information.

```
WinSetErrorInfo (idError, Options, args)
```

Parameters

idError *(ERRORID)* — input
Error identity to be set.

Options *(BIT16)* — input
Options:

SEI_BREAKPOINT	Always enter an INT 3 breakpoint.
SEI_NOBEEP	Do not call DosBeep.
SEI_NOPROMPT	Do not prompt the user with 'Abort, Break or Ignore'.
SEI_STACKTRACE	Save the stack trace.
SEI_REGISTERS	Save the registers.
SEI_ARGCOUNT	The first word in **args** is the count of arguments.
SEI_DOSERROR	The first word in **args** is a DOS error code.

args *(USHORT)* — input
Parameter words.

Remarks

The saved information can be subsequently retrieved by the use of the WinGetLastError and WinGetErrorInfo calls.

This function is only available in the C language and uses the C calling convention to allow a variable number of arguments. The number of arguments depends on the error and its text format as stored in the error message resource segment.

This call requires the existence of a message queue.

WinSetFocus — Set Focus

This call sets the focus window.

WinSetFocus (DeskTop, NewFocus, *Success*)

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

NewFocus *(HWND)* — input
Window handle to receive the focus.

If **NewFocus** identifies a desktop window, no window on the device associated with the **DeskTop** receives the focus.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call is equivalent to the WinFocusChange call in which the **FocusChange** parameter is set to 0.

If no window has the input focus, WM_CHAR messages are posted to the active window's queue, and are not thrown away.

This call requires the existence of a message queue.

WinSetHook – Set Hook

This call installs an application procedure into a specified hook chain.

WinSetHook (hab, hmq, HookType, HookProc, Module, *Success*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

hmq *(HMQ)* – input
Queue identity.

Identifies the queue to which the hook chain belongs. If **hmq** is set to NULL, the hook is installed in the system hook chain. If **hmq** is set to HMQ_CURRENT, the hook is installed in the message queue associated with the current thread (calling thread).

HookType *(SHORT)* – input
Hook chain type.

HK_CODEPAGECHANGE	See CodePageChangeHook.
HK_HELP	See HelpHook.
HK_INPUT	See InputHook.
HK_MSGFILTER	See JournalPlaybackHook.
HK_JOURNALPLAYBACK	See JournalRecordHook.
HK_JOURNALRECORD	See MsgFilterHook.
HK_REGISTERUSERMSG	See RegisterUserMsg.
HK_SENDMSG	See SendMsgHook.

HookProc *(PROC)* – input
Address of the application hook procedure.

Module *(RESID)* – input
Resource identity.

Handle of the module that contains the application hook procedure, as returned by the DosLoadModule or DosGetModHandle call. This parameter can be NULL when a queue hook is being installed by an application into its own message queue.

When hooking a system hook this parameter must be a valid module handle.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HMQ
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

Queue hooks are called before system hooks.

A call to this function installs the hook at the **head** of either the system or queue chain.

The most recently installed hook is called first.

Use WinQueryWindowULong to obtain the queue handle associated with a window handle.

WinSetKeyboardStateTable — Set Keyboard State Table

This call gets or sets the keyboard state.

```
WinSetKeyboardStateTable (DeskTop, KeyStateTable, Set, Success)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

KeyStateTable *(BYTE*256)* — input/output
Key state table.

This is a 256 byte table indexed by virtual key value.

For any virtual key, the 0x80 bit is set if the key is down, and zero if it is up. The 0x01 bit is set if the key is toggled (pressed an odd number of times), otherwise it is zero.

Set *(BOOL)* — input
Set indicator:

TRUE	The keyboard state is set from **KeyStateTable**
FALSE	The keyboard state is copied to **KeyStateTable**.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call does not change the physical state of the keyboard, but changes the value returned by WinGetKeyState, not WinGetPhysKeyState.

To set the state of a single key, first get the entire table, modify the individual key, and then set the table from the modified value.

WinSetMsgInterest — Set Message Interest

SAA

This call sets a window's message interest.

WinSetMsgInterest (hwnd, MsgClass, Control, *Success***)**

Parameters

hwnd *(HWND)* — input
Window handle.

MsgClass *(USHORT)* — input
Message class to have interest level set:

msgid	A single message identity (for example, WM_SHOW)
SMIM_ALL	All messages.

Control *(SHORT)* — input
Interest-identifier for the message class:

SMI_RESET	Revert to interest specified for the window class.
SMI_INTEREST	Interested in the message(s).
SMI_NOINTEREST	Not interested in the message(s).
SMI_AUTODISPATCH	Interested in the message or messages, but they are to be automatically dispatched to the window procedure.

Success *(BOOL)* — return
Interest-changed indicator:

TRUE	Interest successfully changed
FALSE	Interest not successfully changed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call has no effect unless the MsgCtlHook hook, which is invoked by this call, has been set.

This call requires the existence of a message queue.

WinSetMsgMode – Set Message Mode

This call indicates the mode for the generation and processing of messages for the private window class of an application.

WinSetMsgMode (hab, ClassName, Control, *Success*)

Parameters

hab *(HAB)* – input
Anchor block handle.

ClassName *(STRL)* – input
Window class name.

An application specified name for this private class.

Control *(SHORT)* – input
Message mode identifier.

SMD_DELAYED The generation of messages may be delayed
SMD_IMMEDIATE The generation of messages will not be delayed.

Success *(BOOL)* – return
Message delay indicator:

TRUE Message mode successfully set.
FALSE Message mode not successfully set

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call has no effect unless the MsgCtlHook hook, which is invoked by this call, has been set.

This call requires the existence of a message queue.

WinSetMultWindowPos — Set Multiple Window Positions

SAA

This call performs the WinSetWindowPos call for **Count** windows, using **Swp**, an array of structures whose elements correspond to the input parameters of WinSetWindowPos.

WinSetMultWindowPos (hab, Swp, Count, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Swp *(SWP****Count)*** — input
Array.

An array of set window position (SWP) structures. The elements of each correspond to the input parameters of WinSetWindowPos.

Count *(COUNT2)* — input
Window count.

Success *(BOOL)* — return
Positioning success indicator:

TRUE Positioning succeeded
FALSE Positioning failed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG

Remarks

All windows being positioned must have the same parent.

It is more efficient to use this call than to issue multiple WinSetWindowPos calls, as it causes less screen updating. If **hwnd** specifies a frame window, this function recalculates the sizes and positions of the frame controls. If the new window rectangle for any frame control is to be empty, instead of resizing or repositioning that control, it is hidden by (SWP_HIDE) instead. This eliminates needless processing of windows that are not visible. The window rectangle of the control in question is left in its original state. For example, if WinSetWindowPos is issued to change the size of a standard frame window to an empty rectangle, and WinQueryWindowRect is issued against the client window, the rectangle returned is not an empty rectangle, but the original client rectangle before WinSetWindowPos was issued.

This call requires the existence of a message queue.

WinSetOwner — Set Owner

This call changes the owner window of a specified window.

WinSetOwner (hwnd, NewOwner, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle whose owner window is to be changed.

NewOwner *(HWND)* — input
Handle of the new owner:

NULL The window becomes "disowned"
Other Handle of the new owner window.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The old owner window is not locked by this call.

The WinQueryWindow call can be used to get the handle of the owner window.

WinSetParent — Set Parent

SAA

This call sets the parent for **hwnd** to **NewParent**.

WinSetParent (hwnd, NewParent, Redraw, *Success***)**

Parameters

hwnd *(HWND)* — input
Window handle.

NewParent *(HWND)* — input
New parent window handle.

This cannot be a descendant of **hwnd**.

If this parameter is a desktop window handle or HWND_DESKTOP, **hwnd** becomes a main window.

If this parameter is not equal to HWND_OBJECT, it must be a descendant of the same desktop window as **hwnd**.

If this parameter is HWND_OBJECT or a window handle returned by the WinQueryObjectWindow call, **hwnd** becomes an object window.

Redraw *(BOOL)* — input
Redraw indicator:

TRUE If **hwnd** is visible, any necessary redrawing of both the old parent and the new parent windows is performed.
FALSE No redrawing of the old and new parent windows is performed. This avoids an extra device update when subsequent calls cause the windows to be redrawn.

Success *(BOOL)* — return
Parent-changed indicator:

TRUE Parent successfully changed
FALSE Parent not successfully changed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinSetPointer — Set Pointer

This call sets the desktop-pointer handle.

WinSetPointer (DeskTop, NewPointer, *Success***)**

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

NewPointer *(HPOINTER)* — input
New pointer handle:

NULL	Remove pointer from the screen.
Other	Pointer handle associated with **DeskTop**. Handles for application-defined pointers are returned by the WinLoadPointer and WinCreatePointer calls.

Success *(BOOL)* — return
Pointer-updated indicator:

TRUE	Pointer successfully updated
FALSE	Pointer not successfully updated.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_HPTR
PMERR_INV_CURSOR_BITMAP

Remarks

This call is very efficient if **NewPointer** is the same as the current pointer handle.

This call requires the existence of a message queue.

WinSetPointerPos — Set Pointer Position

SAA

This call sets the pointer position.

```
WinSetPointerPos (DeskTop, x, y, Success)
```

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

x *(SHORT)* — input
x position of pointer in screen coordinates.

y *(SHORT)* — input
y position of pointer in screen coordinates.

Success *(BOOL)* — return
Pointer position updated indicator:

TRUE	Pointer position successfully updated
FALSE	Pointer position not successfully updated.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call requires the existence of a message queue.

WinSetPresParam – Set Presentation Parameter

This call sets a presentation parameter for a window.

WinSetPresParam (hwnd, AttrType, AttrValueLen, AttrValue, *Success***)**

Parameters

hwnd *(HWND)* – input
Window handle.

AttrType *(IDENTITY4)* – input
Attribute type identity.

This is either one of the system-defined presentation parameter attribute types (see the **AttrType** parameter of the *PARAM* data type), or an application-defined type.

AttrValueLen *(COUNT4B)* – input
Byte count of the data passed in the **AttrValue** parameter.

AttrValue *(STORAGE)* – input
Attribute value.

See the **AttrValue** parameter of the *PARAM* data type for the values of system-defined attributes.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call associates the presentation parameter attribute identified by **AttrType** with the window **hwnd**. If the attribute already exists for the window, its value is changed to the new value specified by **AttrValue**. If the attribute does not exist, it is added to the window's presentation parameters, with the specified value. (See also WinQueryPresParam and WinRemovePresParam.)

This call requires the existence of a message queue.

WinSetRect — Set Rectangle

This call sets rectangle coordinates.

```
WinSetRect (hab, rect, Left, Bottom, Right, Top, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

rect *(RECT)* — input/output
Rectangle to be updated.

Note: The data type *WRECT* may also be used, if supported by the language.

Left *(SHORT)* — input
Left edge of rectangle.

Bottom *(SHORT)* — input
Bottom edge of rectangle.

Right *(SHORT)* — input
Right edge of rectangle.

Top *(SHORT)* — input
Top edge of rectangle.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call is equivalent to assigning the left, top, right, and bottom arguments to the appropriate fields of *RECT*.

WinSetRectEmpty — Set Rectangle Empty

This call sets a rectangle empty.

```
WinSetRectEmpty (hab, rect, Success)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

rect *(RECT)* — input/output
Rectangle to be set empty.

Note: The data type *WRECT* may also be used, if supported by the language.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call is equivalent to a WinSetRect (**hab**, **rect**, 0, 0, 0, 0) call.

WinSetSynchroMode – Set Synchronization Mode

This call is intended for use in a distributed application.

WinSetSynchroMode (hab, Mode, *Success*)

Parameters

hab *(HAB)* – input
Anchor-block handle.

Mode *(SHORT)* – input
Synchronization mode:

SSM_SYNCHRONOUS	Synchronous mode
SSM_ASYNCHRONOUS	Asynchronous mode
SSM_MIXED	Mixed mode.

Success *(BOOL)* – return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This call allows an application whose message queue is distributed to synchronize the processing of those messages. This is achieved by the use of the MsgCtlHook hook which is invoked by this call.

This call requires the existence of a message queue.

WinSetSysColors — Set System Colors

This call sets system color values.

WinSetSysColors (DeskTop, Options, Format, Start, Tablen, Table, *Success*)

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Options *(BIT32)* — input
Options:

LCOL_RESET
The system colors are all to be reset to default before processing the remainder of the data in this call.
LCOL_PURECOLOR
Color-dithering should not be used to create colors not available in the physical palette. If this option is set, only pure colors are used and no dithering is done.

Format *(ULONG)* — input
Format of entries in the table, as follows:

LCOLF_INDRGB
Array of (index,RGB) values. Each pair of entries is 8 bytes long, comprising 4 bytes for the index, and 4 bytes for the color value. For system color indexes, see **Start**.
LCOLF_CONSECRGB
Array of (RGB) values, corresponding to color indexes **Start** upwards. Each entry is 4 bytes long.

Start *(LONG)* — input
Starting system color index.

This parameter is only applicable if the **Format** parameter is set to LCOLF_CONSECRGB.

The number of system colors (as defined below) is given by SYSCLR_CSYSCOLORS.

The following system color indexes are defined (each successive index is one larger than its predecessor):

SYSCLR_INACTIVETITLETEXT	Inactive title text.
SYSCLR_ACTIVETITLETEXT	Active title text.
SYSCLR_OUTPUTTEXT	Output text.
SYSCLR_WINDOWSTATICTEXT	Static (nonselectable) text.
SYSCLR_SCROLLBAR	Scroll bar halftone area.
SYSCLR_BACKGROUND	Desktop background. Changing this item in any palette changes it in all the palettes.
SYSCLR_ACTIVETITLE	Active window title.
SYSCLR_INACTIVETITLE	Inactive window title.
SYSCLR_MENU	Menu background.
SYSCLR_WINDOW	Window background and slider.
SYSCLR_WINDOWFRAME	Window frame (border line).
SYSCLR_MENUTEXT	Normal menu item text.
SYSCLR_WINDOWTEXT	Instruction text in windows (also the default for static text).
SYSCLR_TITLETEXT	Text in title bar, size box, scroll bar arrow box.
SYSCLR_ACTIVEBORDER	Border fill of active window.
SYSCLR_INACTIVEBORDER	Border fill of inactive window.

WinSetSysColors — Set System Colors

SYSCLR_APPWORKSPACE	Background of specific main windows.
SYSCLR_HELPBACKGROUND	Background of help panels.
SYSCLR_HELPTEXT	Help text.
SYSCLR_HELPHILITE	Highlighted help text.

Tablen *(ULONG)* — input
Number of elements.

Number of elements supplied in **Table**. This may be **0** if, for example, the color table is merely to be reset to the default. For LCOLF_INDRGB it must be an even number.

Table *(LONG*Tablen)* — input
Table.

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 is red intensity value
G is green intensity value
B is blue intensity value.

There are 8 bits for each primary; the maximum intensity for each primary is 255.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

This call sends all main windows in the system a WM_SYSCOLORCHANGE message to indicate that the colors have changed. When this message is received, applications that depend on the system colors can query the new color values with the WinQuerySysColor call.

After the WM_SYSCOLORCHANGE messages are sent, all windows in the system are invalidated so that they are redrawn with the new system colors.

This call does **not** write any system color changes to the initialization file.

This call requires the existence of a message queue.

The following table gives the default RGB values for each color index:

Table 9-1. Color Index Values

System Color Index	Default Color	Default RGB Values
SYSCLR_BACKGROUND [1]	Light gray	204 204 204
SYSCLR_APPWORKSPACE [1]	Off-white [2]	255 251 226
SYSCLR_WINDOW	White	255 255 255
SYSCLR_WINDOWTEXT	Black	0 0 0
SYSCLR_WINDOWSTATICTEXT	Blue	0 0 255
SYSCLR_OUTPUTTEXT	Black	0 0 0
SYSCLR_MENU	White	255 255 255
SYSCLR_MENUTEXT	Black	0 0 0
SYSCLR_ACTIVETITLE [1]	Blue-gray [2]	0 64 128
SYSCLR_INACTIVETITLE [1]	Light gray	204 204 204
SYSCLR_ACTIVETITLETEXT	White	255 255 255
SYSCLR_INACTIVETITLETEXT	Black	0 0 0
SYSCLR_TITLETEXT	White	255 255 255
SYSCLR_ACTIVEBORDER [1]	Yellow	255 255 0
SYSCLR_INACTIVEBORDER [1]	Light gray	204 204 204
SYSCLR_WINDOWFRAME	Dark gray	128 128 128
SYSCLR_SCROLLBAR [1]	Very light gray [2]	223 223 223
SYSCLR_HELPBACKGROUND [1]	White	255 255 255
SYSCLR_HELPTEXT	Dark blue	0 0 127
SYSCLR_HELPHILITE	Dark green	0 127 127
SYSCLR_HILITEFOREGROUND	White	255 255 255
SYSCLR_HILITEBACKGROUND [1]	Blue-gray [2]	0 64 128
SYSCLR_BUTTONLIGHT	White	255 255 255
SYSCLR_BUTTONMIDDLE	Light gray	204 204 204
SYSCLR_BUTTONDARK	Dark gray	128 128 128
SYSCLR_BUTTONDEFAULT	Black	0 0 0
SYSCLR_TITLEBOTTOM	Light gray	204 204 204
SYSCLR_SHADOW	Light gray	204 204 204
SYSCLR_ICONTEXT	Black	0 0 0
SYSCLR_DIALOGBACKGROUND	White	255 255 255

Notes:

1. Dithered colors allowed for this index.
2. For the Enhanced Graphics Adapter and the Video Graphics Adapter, this color is produced by color dithering.

WinSetSysModalWindow — Set System Modal Window

SAA

This call makes a window become the system-modal window, or ends the system-modal state.

```
WinSetSysModalWindow (Desktop, hwnd, Success)
```

Parameters

Desktop *(HWND)* — input
Desktop-window handle, or HWND_DESKTOP.

hwnd *(HWND)* — input
Handle of window to become system-modal window.

If NULL, system-modal state is ended, and input processing returns to its normal state.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

Input processing can enter a "system modal" state. In this state, all pointing device and keyboard input is directed to a special window, known as the system-modal window, or to one of its child windows (or a window owned by one of them). An "owned" window is a window that refers to its owner window set by using either the **Owner** parameter of the WinCreateWindow function or the **NewOwner** parameter of the WinSetOwner function. All other main windows behave as though they are disabled and no interaction is possible with them.

Programming Note: The disabled windows are not actually disabled, but made noninteractive. No messages are sent to these windows when the system-modal state is entered or left, and their WS_DISABLE style bits are not changed.

Where a system-modal window exists and another window is explicitly made the active window, the newly activated window becomes the system-modal window. This replaces the old one, which becomes a noninteractive window. When the system-modal window is destroyed, the system-modal state is ended, and input processing returns to its normal state.

This call should only be called while processing pointing device or keyboard input.

The new system-modal window is **not** locked during the processing of this call.

This call requires the existence of a message queue.

WinSetSysValue — Set System Value

This call sets the system value.

WinSetSysValue (DeskTop, ValueId, Value, *Success***)**

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	Set the system values for the desktop-window handle.
Other	Set the system values for the specified desktop-window handle.

ValueId *(SHORT)* — input
System-value identity.

The following values are settable:

SV_CXSIZEBORDER	Width of sizing border
SV_CYSIZEBORDER	Height of sizing border
SV_SWAPBUTTON	If this value is TRUE, the mouse buttons are set for left-handed use
SV_CURSORRATE	Cursor blink rate in milliseconds
SV_DBLCLKTIME	Mouse double-click time in milliseconds
SV_CXDBLCLK	Width of mouse double-click sensitive area
SV_CYDBLCLK	Height of mouse double-click sensitive area
SV_ALARM	TRUE if alarm sound is ENABLED; FALSE turns off the alarm sound
SV_WARNINGFREQ	Frequency for warning alarms
SV_WARNINGDURATION	Duration for warning alarms
SV_NOTEFREQ	Frequency for note alarms
SV_NOTEDURATION	Duration for note alarms
SV_ERRORFREQ	Frequency for error alarms
SV_ERRORDURATION	Duration for error alarms
SV_FIRSTSCROLLRATE	Delay in milliseconds, before autoscrolling starts, when using a scroll bar
SV_SCROLLRATE	Delay in milliseconds, between scroll operations, when using a scroll bar
SV_SETLIGHTS	When TRUE, the appropriate light is set when the keyboard state table is set.

Value *(LONG)* — input
System value.

Success *(BOOL)* — return
Value-set indicator:

TRUE	System value successfully set
FALSE	System value not successfully set.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

WinSetSysValue — Set System Value

SAA

Remarks

Valueid must be a valid SV_* value (see page 9-224).

WinSetValue — Set Value

This call creates a 4-byte integer composed of 1-byte or 2-byte integers or bit values.

WinSetValue (hab, Count, Types, Data, Value, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Count *(LONG)* — input
Number of elements.

Must be greater than zero, and not more than the number of values that can be accommodated in the **Value** parameter, under the control of the **Types** parameter.

Types *(LONG****Count)** — input
Data types of the components of the value to be composed:

DTYP_BIT8	8-bit value
DTYP_BIT16	16-bit value
DTYP_BYTE	1-byte unsigned integer
DTYP_UCHAR	1-byte unsigned integer
DTYP_SHORT	2-byte signed integer
DTYP_USHORT	2-byte unsigned integer.

Data *(LONG****Count)** — input
Data components of the value to be composed.

Value *(LONG)* — output
4-byte integer.

Success *(BOOL)* — return
Success indicator:

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError:

PMERR_INV_HAB
PMERR_ARRAY_TOO_SMALL
PMERR_ARRAY_TOO_LARGE
PMERR_DATATYPE_ENTRY_INVALID

Remarks

The elements of the **Data** parameter are processed in order, so that the first element sets the most significant byte(s) within the **Value** parameter, and the last element sets the least significant byte(s). During the creation of the **Value** parameter, bytes may be skipped to ensure that the components have the correct alignment; skipped bytes are set to null. If fewer data elements are specified than are needed to fill the **Value** parameter, the remaining bytes are set to null.

Values whose data types are DTYP_BIT8 or DTYP_BIT16 must occupy the least significant byte(s) in the elements of the **Data** parameter.

Programming Note: This function is required only in COBOL and FORTRAN.

This call requires the existence of a message queue.

WinSetWindowBits — Set Window Word Bits

This call sets a number of bits into the memory of the reserved window words.

WinSetWindowBits (hwnd, b, Data, Mask, *Success***)**

Parameters

hwnd *(HWND)* — input
Window handle.

b *(SHORT)* — input
Zero-based index of the value to be set.

The units of **b** are bytes. Valid values are zero through (**Extra** —4), where **Extra** is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. Any of the QWL_* values specified for the WinQueryWindowULong call can also be used.

Data *(BIT32)* — input
Bit data to store in the window words.

This is done under the control of the **Mask** parameter.

Mask *(BIT32)* — input
Bits to be written indicator.

A "1" bit indicates that the corresponding bit of the **Data** parameter is to be stored into the window word. A "0" bit indicates that the corresponding bit of the **Data** parameter is to be ignored in the storing operation; the value of that bit position in the window word is unaltered.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

The bits are set in a single operation.

This call requires the existence of a message queue.

WinSetWindowPos — Set Window Position

This call allows the general positioning of a window.

WinSetWindowPos (hwnd, Behind, x, y, cx, cy, Options, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle.

Behind *(HWND)* — input
Relative window-placement order.

Ignored if SWP_ZORDER is not selected. Values that can be specified are:

HWND_TOP	Place **hwnd** on top of all siblings
HWND_BOTTOM	Place **hwnd** behind all siblings
Other	Identifies the sibling window behind which **hwnd** is to be placed.

x *(SHORT)* — input
Window position, x coordinate.

This is the x coordinate of **hwnd**. It is in window coordinates relative to the bottom left corner of its parent, but is ignored if SWP_MOVE is not selected.

y *(SHORT)* — input
Window position, y coordinate.

This is the y coordinate of **hwnd**. It is in window coordinates relative to the bottom left corner of its parent, but is ignored if SWP_MOVE is not selected.

cx *(SHORT)* — input
Window size.

This specifies the width of **hwnd** in device units, but is ignored if SWP_SIZE is not selected.

cy *(SHORT)* — input
Window size.

This specifies the depth of **hwnd** in device units, but is ignored if SWP_SIZE is not selected.

Options *(BIT16)* — input
Window-positioning options.

One or more of these options can be specified:

SWP_SIZE	Change the window size.
SWP_MOVE	Change the window x,y position.
SWP_ZORDER	Change the relative window placement.
SWP_SHOW	Show the window.
SWP_HIDE	Hide the window.
SWP_NOREDRAW	Changes are not redrawn.
SWP_NOADJUST	Do not send a WM_ADJUSTWINDOWPOS message before moving or sizing.
SWP_ACTIVATE	Activate the **hwnd** window if it is a frame window. This indicator has no effect on other windows. The frame window is made the topmost window, unless SWP_ZORDER is specified also in which instance the **Behind** window is used.

SWP_DEACTIVATE Deactivate the **hwnd** window if it is a frame window. This indicator has no effect on other windows.

The frame window is made the bottommost window, unless SWP_ZORDER is specified, in which instance the **Behind** window is used.

SWP_MINIMIZE Minimize the window. This indicator has no effect if the window is in a minimized state, and is also mutually exclusive with SWP_MAXIMIZE and SWP_RESTORE.

SWP_MAXIMIZE Maximize the window. This indicator has no effect if the window is in a maximized state, and is also mutually exclusive with SWP_MINIMIZE and SWP_RESTORE.

SWP_RESTORE Restore the window. This indicator has no effect if the window is in its normal state, and is also mutually exclusive with SWP_MINIMIZE and SWP_MAXIMIZE.

The position and size of the window in its normal state is remembered in its window words when it is first maximized or minimized, although these values can be altered by use of the WinSetWindowUShort call.

The window is restored to the position and size remembered in its window words, unless the SWP_MOVE or SWP_SIZE indicators are set. These indicators cause the position and size values specified in this call to be used.

Success *(BOOL)* — return
Repositioning indicator:

TRUE Window successfully repositioned
FALSE Window not successfully repositioned.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_FLAG

Remarks

Programming Note: Messages may be received from other processes or threads during the processing of this call.

If a window created with the CS_SAVEBITS style is reduced, the screen image saved is used to redraw the area uncovered when the window size changes, if those bits are still valid.

If the CS_SIZEREDRAW style is present, the entire window area is assumed invalid if sized. Otherwise, WM_CALCVALIDRECTS is sent to the window to inform the window manager which bits it may be possible to preserve.

Messages sent from WinSetWindowPos and WinSetMultWindowPos have specific orderings within the window-positioning process. The process begins with redundancy checks and precalculations on every window for each requested operation. For example, if SWP_SHOW is present but the window is already visible, SWP_SHOW is turned off. If SWP_SIZE is present, and the new size is equal to the old size, SWP_SIZE is turned off.

If the operations create new results, the information is calculated and stored (for instance, when sizing or moving, the new window rectangle is stored for later use). It is at this point that the WM_ADJUSTWINDOWPOS message is sent to any window that is sizing or moving. It is also at this point that the WM_CALCVALIDRECTS message is sent to any window that is sizing and does not have the CS_SIZEREDRAW window style.

When all the new window states are calculated, the window-management process begins. Window areas that can be preserved are moved from the old to the new positions, window areas that are invalidated by these operations are calculated and distributed as update regions. When this is finished, and before any synchronous-paint windows are repainted, the WM_SIZE message is sent to any windows that have changed size. Next, all the synchronous-paint windows that can be are repainted, and the process is complete.

If a synchronous-paint parent window has a size-sensitive area displayed that includes synchronous-paint child windows, the parent needs to reposition those windows when it receives the WM_SIZE message. Their invalid regions are added to the parent's invalid region, resulting in one update after the parent's WM_SIZE message, rather than many independent (and later, duplicated) updates.

Programming Note: Certain windows will not be positioned precisely to the parameters of this function, but according to the behavior of their window procedure. For example, frame windows without a style creation flag of FCF_NOBYTEALIGN will not position to any specific screen coordinate. Similarly, frame windows with zero size and position are created by the WinCreateStdWindow function and therefore these values are treated as a special case by the frame window procedure.

Messages sent by this function are:

WM_CALCVALIDRECTS	Sent to determine the area of a window that may be possible to preserve as the window is sized.
WM_SIZE	Sent if the size of the window has changed, after the change has been made.
WM_MOVE	Sent when a window with CS_MOVENOTIFY class style moves its absolute position.
WM_ACTIVATE	Sent if a different window becomes the active window. See also WinSetActiveWindow.
WM_ADJUSTWINDOWPOS	Not sent if SWP_NOADJUST is specified. The message contains an SWP structure that has been filled in by this call with the proposed move/size data. The window can adjust this new position by changing the contents of the *SWP* structure. If **hwnd** specifies a frame window, this function recalculates the sizes and positions of the frame controls. If the new window rectangle for any frame control is empty, instead of resizing or repositioning that control, it is hidden if SWP_HIDE is specified. This eliminates needless processing of windows that are not visible. The window rectangle of the control in question is left in its original state. For example, if WinSetWindowPos is issued to change the size of a standard-frame window to an empty rectangle, and WinQueryWindowRect is issued against the client window, the rectangle returned is not an empty rectangle, but the original client rectangle before WinSetWindowPos was issued.

This call requires the existence of a message queue.

WinSetWindowPtr — Set Window Words Pointer

This call sets a pointer value into the memory of the reserved window words.

```
WinSetWindowPtr (hwnd, b, p, Success)
```

Parameters

hwnd *(HWND)* — input
Window handle.

b *(SHORT)* — input
Zero-based index into the window words.

The units of b are bytes. Valid values are zero through (**Extra** —4), where **Extra** is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage.

The value QWP_PFNWP can be used as the index for the address of the window procedure for the window.

p *(STORAGE)* — input
Pointer value to store in the window words.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

Remarks

This call requires the existence of a message queue.

WinSetWindowText — Set Window Text

This call sets the window text for **hwnd** to **String**.

WinSetWindowText (hwnd, String, *Result*)

Parameters

hwnd *(HWND)* — input
Window handle.

String *(STRL)* — input
Window text.

Result *(BOOL)* — return
Text-updated indicator:

TRUE Text successfully updated
FALSE Text was not successfully updated.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call sends a WM_SETWINDOWPARAMS message to **hwnd**.

If this call references the window of another process, **String** must be in memory that is shared by both processes, or a memory fault may occur.

If **hwnd** has a style of WS_FRAME, the title-bar window text is set.

Some window classes interpret the **String** in a special way. The tilde character (~) indicates that the following character is a mnemonic; for details, see Chapter 13, "Button Control Window Processing" and Chapter 17, "Menu Control Window Processing."

This call requires the existence of a message queue.

WinSetWindowULong — Set Window Word Long

This call sets an unsigned, long integer value into the memory of the reserved window words.

```
WinSetWindowULong (hwnd, b, Data, Success)
```

Parameters

hwnd *(HWND)* — input
Window handle.

b *(SHORT)* — input
Zero-based index of the value to be set.

The units of **b** are bytes. Valid values are zero through (**Extra** —4), where **Extra** is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. So too are any of the QWL_* values, as specified for the WinQueryWindowULong call.

QWS_* values can not be used.

Data *(ULONG)* — input
Unsigned, long integer value to store in the window words.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_PARAMETER_OUT_OF_RANGE

WinSetWindowUShort — Set Window Word Short

This call sets an unsigned, short integer value into the memory of the reserved window words.

```
WinSetWindowUShort (hwnd, b, Data, Success)
```

Parameters

hwnd *(HWND)* — input
Window handle.

b *(SHORT)* — input
Zero-based index of the value to be set.

The units of **b** are bytes. Valid values are zero through (**Extra** –2), where **Extra** is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. So too are any of the QWS_* values, as specified for the WinQueryWindowUShort call.

QWL_* values cannot be used.

Data *(USHORT)* — input
Unsigned, short integer value to store in the window words.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

WinShowCursor — Show Cursor

SAA

This call shows or hides the cursor that is associated with a specified window.

WinShowCursor (hwnd, Show, *Success*)

Parameters

hwnd *(HWND)* — input
Handle of window to which the cursor belongs.

Show *(BOOL)* — input
Show indicator:

TRUE Make cursor visible
FALSE Make cursor invisible.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred, or an attempt was made to show the cursor when it was already visible.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This function must be called by the same thread that created the cursor that is affected.

A cursor show-level count is maintained. It is incremented by a hide operation and decremented by a show operation. The cursor is actually visible if the cursor show-level count is zero, otherwise it is invisible. When decrementing, the cursor show-level count is fixed at zero so as not to show the cursor too many times, but it is possible to hide the cursor a number of times in succession.

This call requires the existence of a message queue.

WinShowPointer — Show Pointer

This call adjusts the pointer display level to show or hide a pointer.

WinShowPointer (DeskTop, Show, *Success***)**

Parameters

DeskTop *(HWND)* — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	The specified desktop-window handle.

Show *(BOOL)* — input
Level-update indicator:

TRUE	Decrement pointer display level by one. (The pointer level is not decremented to a negative value.)
FALSE	Increment pointer display level by one.

Success *(BOOL)* — return
Display-level-updated indicator:

TRUE	Pointer display level successfully updated
FALSE	Pointer display level not successfully updated.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The pointer display level determines whether the pointer is shown. If it is zero, the pointer is visible, but if it is greater than zero, the pointer is not visible. The initial setting of the pointer display level is dependent on the capabilities of the device. If a pointing device exists, the initial setting of the pointer display level is zero, otherwise it is one. The existing pointer display level can be obtained by using the WinQuerySysValue call with **ValueId** set to SV_POINTERLEVEL.

This call requires the existence of a message queue.

WinShowTrackRect – Show Tracking Rectangle

This call hides or shows the tracking rectangle.

```
WinShowTrackRect (hwnd, Show, Success)
```

Parameters

hwnd *(HWND)* – input
Window handle.

Passed to the WinTrackRect function.

Show *(BOOL)* – input
Show indicator:

TRUE Show the tracking rectangle
FALSE Hide the tracking rectangle.

Success *(BOOL)* – return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call maintains a show count. When a hide request is made, this count is decremented; when a show request is made, the 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.

When a rectangle is tracking, the application must call this function to hide the rectangle if there is a possibility of corrupting the tracking rectangle while drawing. The rectangle is shown afterwards. Since the **track** structure is updated continuously, the application can examine the coordinates of the current tracking rectangle to determine whether temporary hiding is necessary.

The only case where an application needs to use this call is during asynchronous drawing. If an application is drawing on one thread, and issuing WinTrackRect on another, unwanted areas of tracking rectangle may be left behind. The drawing thread is therefore responsible for calling this function whenever tracking is in progress. The application must provide for communication between the two threads to ensure that if one thread is tracking, the drawing thread issues this call. This can be achieved with a *semaphore*.

This call requires the existence of a message queue.

WinShowWindow — Show Window

This call sets the visibility state of a window.

WinShowWindow (hwnd, NewVisibility, *Success*)

Parameters

hwnd *(HWND)* — input
Window handle.

NewVisibility *(BOOL)* — input
New visibility state:

TRUE Set window state visible
FALSE Set window state invisible.

Success *(BOOL)* — return
Visibility changed indicator:

TRUE Window visibility successfully changed
FALSE Window visibility not successfully changed.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

A window possesses a visibility state indicated by the WS_VISIBLE style bit. When the WS_VISIBLE style bit is set, the window is shown and subsequent drawing into the window is presented, unless that window is obscured by some other window, or at least one of the windows upwards in the parent chain from **hwnd** does not have the WS_VISIBLE style.

When the WS_VISIBLE style bit is not set, the window is not shown ("hidden") and subsequent drawing into the window is not presented, even if that window is not obscured by another window.

If the value of the WS_VISIBLE style bit has been changed, the WM_SHOW message is sent to the window of **hwnd** before the function returns.

This call requires the existence of a message queue.

WinStartTimer – Start Timer

This call starts a timer.

```
WinStartTimer (hab, hwnd, Timer, Timeout, Ret)
```

Parameters

hab *(HAB)* – input
Anchor-block handle.

hwnd *(HWND)* – input
Window handle that is part of the timer identification.

NULL The **Timer** parameter is ignored, and this call returns a unique, nonzero, identity which represents that timer. The timer message is posted in the queue associated with the current thread, with the **hwnd** parameter of the *QMSG* structure set to NULL.
Other Window handle.

Timer *(IDENTITY)* – input
Timer identifier.

The value of an application-timer identifier must be below TID_USERMAX to avoid clashes with timers used by the system.

A timer identification, TID_SCROLL, is created by a scroll bar control. An application does not normally see the associated WM_TIMER, but passes it to the scroll-bar control.

A timer identification, TID_CURSOR, is created when the cursor is flashing. An application must ensure that the associated WM_TIMER is passed on to the default window procedure.

Timeout *(USHORT)* – input
Delay time in milliseconds.

Ret *(USHORT)* – return
Return code.

When **hwnd** is set to NULL:

0 Error occurred
Other Timer identity.

When **hwnd** is set to other than NULL:

0 Error occurred
Other Successful completion.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call creates a timer identified by **hwnd** and **Timer**, set to time out every **Timeout** milliseconds. When a timer times out, a WM_TIMER message is posted.

A **Timeout** value of zero causes the timer to timeout as fast as possible; generally, this is about 1/18th of a second.

A second call to this function, for a timer that already exists, resets that timer.

This call requires the existence of a message queue.

WinStopTimer — Stop Timer

This call stops a timer.

WinStopTimer (hab, hwnd, Timer, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

hwnd *(HWND)* — input
Window handle.

Timer *(USHORT)* — input
Timer identifier.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred, or timer did not exist.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

When this function is called, no further messages are received from the stopped timer, even if it has timed out since the last call to WinGetMsg.

This call requires the existence of a message queue.

WinSubclassWindow — Subclass Window

This call subclasses the indicated window by replacing its window procedure with another window procedure, specified by **NewWindowProc**.

WinSubclassWindow (hwnd, NewWindowProc, *OldWindowProc***)**

Parameters

hwnd *(HWND)* — input
Handle of window that is being subclassed.

NewWindowProc *(WNDPROC)* — input
New window procedure.

Window procedure used to subclass **hwnd**.

OldWindowProc *(WNDPROC)* — return
Old window procedure.

Previous window procedure belonging to **hwnd**.

If this function fails, **0L** is returned.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

To subclass a window effectively, the new window procedure calls the old window procedure rather than WinDefWindowProc, for those messages it does not process itself.

To reverse the effect of subclassing, call this function again using the old window procedure address.

Note: It is not possible to subclass a window created by another process.

This call requires the existence of a message queue.

WinSubstituteStrings — Substitute Strings

This call performs a substitution process on a text string, replacing specific marker characters with text supplied by the application.

WinSubstituteStrings (hwnd, Src, DestMax, Dest, *DestRet*)

Parameters

hwnd *(HWND)* — input
Handle of window that processes the call.

Src *(STRL)* — input
Source string.

This is the text string that is to have substitution performed.

DestMax *(SHORT)* — input
Maximum number of characters returnable.

This is the maximum number of characters that can be returned in **Dest**.

Dest *(STRL)* — output
Resultant string.

This is the text string produced by the substitution process.

The string is truncated if it would otherwise contain more than **DestMax** characters. When truncation occurs, the last character of the truncated string is always the null-termination character.

DestRet *(SHORT)* — return
Actual number of characters returned.

This is the actual number returned in **Dest**, excluding the null-termination character. The maximum value is (**DestMax**—1). It is zero if an error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

When a string of the form "%n" (where n is in the range 0 through 9) occurs in the source string, a WM_SUBSTITUTESTRING message is sent to the specified window. This message returns a text string to use as a substitution for "%n" in the destination string, which is otherwise an exact copy of the source string.

If "%%" occurs in the source, "%" is copied to the destination, but no other substitution occurs. If "%x" occurs in the source, where x is not a digit or "%," the source is copied unchanged to the destination. The source and destination strings must not overlap in memory.

This call is particularly useful for displaying variable information in dialogs, menus, and other user-interface calls. Variable information can include such things as file names, which cannot be statically declared within resource files.

This function is called by the system while creating child windows in a dialog box. It allows the child windows to perform textual substitutions in their window text.

This call requires the existence of a message queue.

WinSubtractRect — Subtract Rectangle

This call subtracts one rectangle from another.

WInSubtractRect (hab, Dest, Src1, Src2, *Nonempty*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

Dest *(RECT)* — output
Result.

The result of the subtraction of **Src2** from **Src1**.

Note: The data type *WRECT* can also be used, if supported by the language.

Src1 *(RECT)* — input
First source rectangle.

Note: The data type *WRECT* can also be used, if supported by the language.

Src2 *(RECT)* — input
Second source rectangle.

Note: The data type *WRECT* can also be used, if supported by the language.

Nonempty *(BOOL)* — return
Not-empty indicator:

TRUE Rectangle is not empty
FALSE Rectangle is empty or an error occurred.

Remarks

Subtracts **Src2** from **Src1**. **Src1**, **Src2**, and **Dest** must be distinct *RECT* structures.

Subtracting one rectangle from another does not always result in a rectangular area. When this occurs, this call returns **Src1** in **Dest**. For this reason, this call provides only an approximation of subtraction. However, the area described by **Dest** is always greater than, or equal to, the true result of the subtraction.

The GpiCombineRegion call can be used to calculate the true result of the subtraction of two rectangular areas. The WinSubtractRect call is much faster.

WinSwitchToProgram — Switch To Program

This call makes a specific program the active program.

```
WinSwitchToProgram (SwHandle, RetCode)
```

Parameters

SwHandle *(HSWITCH)* — input
Task-list entry handle of program to be activated.

RetCode *(USHORT)* — return
Return code.

0	Successful completion.
INV_SWITCH_LIST_ENTRY_HANDLE	Invalid task-list entry handle of the program to be activated.
NOT_PERMITTED_TO_CAUSE_SWITCH	Requesting program is not the current foreground process.

Remarks

Use of this function causes another window (and its related windows) of a Presentation Manager screen group to appear on the front of the screen, or a switch to another screen group in the case of a non-Presentation Manager program. In either case, the keyboard (and mouse for the non-Presentation Manager case) input is directed to the new program.

A program can only be made the foreground process by the application which is the current foreground process. This call is ignored if the issuer is not the current foreground process.

A foreground process is defined as being any process within the active non-Presentation Manager screen group, or the window with the input focus for a Presentation Manager screen group.

This call requires the existence of a message queue.

WinTerminate — Terminate

SAA

This call terminates an application thread's use of the Presentation Manager and releases all of its associated resources.

```
WinTerminate (hab, Terminated)
```

Parameters

hab *(HAB)* — input
Anchor-block handle.

Terminated *(BOOL)* — return
Termination indicator:

TRUE	Application usage of Presentation Manager successfully terminated
FALSE	Application usage of Presentation Manager not successfully terminated, or WinInitialize has not been issued on this thread.

Remarks

It is good practice to issue this call before terminating an application thread. Before issuing this call, the application should destroy all windows and message queues that have been created by the thread, and return any cached presentation spaces to the cache. If it does not do so, the results, and the return value from this and subsequent calls are indeterminate.

This call requires the existence of a message queue.

WinTerminateApp – Terminate Application

This call terminates an application.

```
WinTerminateApp (happ, Success)
```

Parameters

happ *(HAPP)* – input
Application handle.

Returned by the WinInstStartApp call.

Success *(BOOL)* – return
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HAPP
PMERR_CANNOT_STOP

Remarks

Terminates the application identified by the **happ** parameter, which has been started using a WinInstStartApp call, whose **Options** parameter was set with the SAF_STARTCHILDAPP value.

The specified application may refuse to stop, in which case this call still returns TRUE. The only way to ensure that the application has terminated is to wait for the appropriate WM_APPTERMINATENOTIFY message to be posted to the window handle specified on the **NotifyWindow** parameter of the WinInstStartApp call.

This call requires the existence of a message queue.

WinThrow —
Restore Execution Environment

This function restores the execution environment to a previously saved state.

```
WinThrow (CatchBuf, ThrowBack)
```

Parameters

CatchBuf *(CATCHBUF)* — input
Saved execution-environment buffer.

ThrowBack *(SHORT)* — input
Return value.

This value becomes the value of the **RetCode** parameter of the WinCatch function, which saves the execution environment in the **CatchBuf** parameter specified by this function.

Remarks

This function is the complement of the WinCatch function, which saves the execution environment restored by this function.

Note: This function does not return.

This call requires the existence of a message queue.

WinTrackRect – Draw Tracking Rectangle

This call draws a tracking rectangle.

WinTrackRect (hwnd, hps, TrackInfo, *Success*)

Parameters

hwnd *(HWND)* – input
Window handle where tracking is to take place.

It is assumed that the style of this window is not WS_CLIPCHILDREN.

HWND_DESKTOP Track over the entire screen
Other Track over specified window only.

hps *(HPS)* – input
Presentation-space handle.

Used for drawing the clipping rectangle:

NULL The **hwnd** parameter is used to calculate a presentation space for tracking. It is assumed that tracking takes place within **hwnd** and that the style of this window is not WS_CLIPCHILDREN. Thus, when the drag rectangle appears, it is not clipped by any children within the window. If the window style is WS_CLIPCHILDREN and the application wants the drag rectangle to be clipped, it must explicitly pass an appropriate presentation space.
Other Specified presentation-space handle.

TrackInfo *(TRACKINFO)* – input/output
Track information.

Success *(BOOL)* – return
Success indicator:

TRUE Tracking successful.
FALSE Tracking canceled, or the pointing device was already captured when this function was called.

Only one tracking rectangle can be in use at one time.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call provides general pointing device tracking. It draws a rectangle and enables the user to position the entire rectangle, or size a specific side or corner, as required. The resulting 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 by means of the wide sizing borders simply calls this function.

This function determines which button of the pointing device is depressed at the start of the operation (that is, the left or the right button) and only completes the tracking operation when the same button is released.

WinTrackRect – Draw Tracking Rectangle

This call enables the caller to control such limiting values as:

- A maximum and minimum tracking size
- Absolute tracking-position limits
- The tracking rectangle side widths
- A restriction of tracking rectangle movements to a predefined positional grid.

If the **hwnd** parameter is equal to HWND_DESKTOP, then this call automatically calls WinLockWindowUpdate to prevent screen output while tracking. When tracking has been completed, the screen is unlocked before this call returns. For other windows, it is the responsibility of the calling application to prevent the update of the screen during tracking.

If the **options** parameter of the *TRACKINFO* structure specified by the TF_SETPOINTERPOS value is included, the pointing device pointer is positioned at the center of the tracking rectangle. Otherwise, the pointing device pointer is not moved from its current position and delta is established between the pointing device position and the part of the tracking rectangle that it moves (the value of which is kept constant).

While moving or sizing with the keyboard interface, the pointing device 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 cursor	Moves the pointing device pointer and tracking rectangle left.
Up cursor	Moves the pointing device pointer and tracking rectangle up.
Right cursor	Moves the pointing device pointer and tracking rectangle right.
Down cursor	Moves the pointing device pointer and tracking rectangle down.
Esc	Cancels the current tracking operation. In this instance, the value of the tracking rectangle is undefined on exit.

The pointing device and the keyboard interface can be intermixed. The caller need not include the TF_SETPOINTERPOS value to be able to use the keyboard interface, as this value simply initializes the position of the pointing device pointer.

Programming Note: Tracking movements using the keyboard arrow keys, move in increments of the values of the **gridwidth** and **gridheight** parameters of the *TRACKINFO* structure specified by the **Trackinfo** parameter, regardless of whether TF_GRID is specified.

If TF_GRID is specified, the interior of the tracking rectangle is only allowed in multiples of the values of the **gridwidth** and **gridheight** parameters. The default values for these are the system font character width and half the system font character height, respectively.

The tracking rectangle is usually logically "on top" of objects it tracks , so that the user can see the old size and position while tracking the new. Thus, it is possible for a window "below" the tracking rectangle to be updated while part of the tracking rectangle is "above" it.

Because the tracking rectangle is drawn in exclusive-OR mode, no window can draw below the tracking rectangle (and thereby obliterate it) without first notifying the tracking code, because unwanted areas of the tracking rectangle can be left behind. If the window doing the drawing is clipped out from the window in which the tracking is occurring, this problem does not arise.

WinTrackRect — Draw Tracking Rectangle

To prevent a window that is currently processing a WM_PAINT message drawing over the tracking rectangle, OS/2 Version 1.2 treats the tracking rectangle as a system-wide resource, only one of which can be in use at any time. If there is a risk of the currently updating window drawing on the tracking rectangle, OS/2 Version 1.2 removes the tracking rectangle while that window and its children update, and then replaces it. This is done during the WinBeginPaint and WinEndPaint functions respectively. If the tracking rectangle overlaps, it is removed in the WinBeginPaint call. In the WinEndPaint call all the children are updated by means of the WinUpdateWindow call before the tracking rectangle is redrawn.

This call has a modal loop within it. The loop has a HK_MSGFILTER hook and a MSGF_TRACK hook code.

Programming Note: It is ensured that the rectangle tracked by this function is within the specified tracking bounds and dimensions. If the rectangle passed is out of these bounds, or too large or too small, it is modified to a rectangle that meets those limits.

This call requires the existence of a message queue.

WinTranslateAccel — Translate Accelerator

SAA

This call translates a WM_CHAR message.

WinTranslateAccel (hab, hwnd, Accel, Qmsg, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

hwnd *(HWND)* — input
Destination window.

Accel *(HACCEL)* — input
Accelerator-table handle.

Qmsg *(QMSG)* — input/output
Message to be translated.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_INVALID_HACCEL

Remarks

This call translates **Qmsg** if it is a WM_CHAR message in the accelerator table **Accel**. The message is translated into a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message, with **hwnd** identifying the destination window. Normally, this parameter is a frame-window handle. This call does not highlight menu items.

If **Accel** equals NULL, the current accelerator table is assumed.

WinTranslateAccel returns TRUE if the message matches an accelerator in the table. **Qmsg** is modified by WinTranslateAccel if a match is found.

If a menu item exists that matches the accelerator-command value, and that item is disabled, **Qmsg** is translated to a WM_NULL message, rather than a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message. If the command is WM_SYSCOMMAND or WM_HELP (and if a WM_SYSCOMMAND or FID_SYSMENU child window is searched) the menu child window of **hwnd** that has the FID_MENU identifier is searched.

It is possible to have accelerators that do not correspond to items in a menu. If the command value does not match any items in the menu, the message is still translated.

Generally, applications do not have to call this function; it is usually called automatically by WinGetMsg and WinPeekMsg, when a WM_CHAR message is received with the window handle of the active window as the first parameter. The standard frame window procedure always passes WM_COMMAND messages to the FID_CLIENT window. Because the message is physically changed by WinTranslateAccel, applications do not see the WM_CHAR messages that result in WM_COMMAND, WM_SYSCOMMAND, or WM_HELP messages.

This call requires the existence of a message queue.

WinUnionRect — Union Rectangle

This call calculates a rectangle that bounds the two source rectangles.

WinUnionRect (hab, Dest, Src1, Src2, *Nonempty***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Dest *(RECT)* — output
Bounding rectangle.

Note: The data type *WRECT* can also be used, if supported by the language.

Src1 *(RECT)* — input
First source rectangle.

Note: The data type *WRECT* can also be used, if supported by the language.

Src2 *(RECT)* — input
Second source rectangle.

Note: The data type *WRECT* can also be used, if supported by the language.

Nonempty *(BOOL)* — return
Non-empty indicator:

TRUE **Dest** is a non-empty rectangle
FALSE Error, or **Dest** is an empty rectangle.

Remarks

Src1 and **Src2** must not be NULL pointers, although the rectangles they point to can be empty (see the WinIsRectEmpty call).

If one of the source rectangles is empty, the other is returned.

WinUpdateWindow — Update Window

SAA

This call forces the update of a window and its associated child windows.

```
WinUpdateWindow (hwnd, Success)
```

Parameters

hwnd *(HWND)* — input
Window handle.

Success *(BOOL)* — return
Window-updated indicator:

TRUE Window successfully updated
FALSE Window not successfully updated.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

If **hwnd** is an asynchronous window, only it and its asynchronous children are updated. They are sent WM_PAINT messages from this call.

If **hwnd** is a synchronous window, only it and its synchronous children are updated. They are sent WM_PAINT messages from this call. If the window is owned by a different thread from the thread issuing the call, the message is sent asynchronously and not synchronously.

If **hwnd** is a child of a nonclip-children parent, the update region of **hwnd** is subtracted from the update region of the parent, if the parent has one. This is so that any parent-window drawing after **hwnd** does not draw over whatever is drawn by **hwnd**.

This call requires the existence of a message queue.

WinUpper — Uppercase String

This call converts a string to uppercase.

WinUpper (hab, Codepage, Country, String, *RetLen***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Codepage *(USHORT)* — input
Code page:

NULL Use the current-process code page
Other Use the specified code page.

Country *(USHORT)* — input
Country code:

NULL Use the default country code specified in CONFIG.SYS
Other Use the specified country code.

String *(STRL)* — input/output
String to be converted to uppercase.

RetLen *(USHORT)* — return
Length of converted string.

Possible returns from WinGetLastError:

PMERR_INVALID_STRING_PARM

Remarks

This call requires the existence of a message queue.

WinUpperChar — Uppercase Character

This call translates a character to uppercase.

WinUpperChar (hab, Codepage, Country, Inchar, *Outchar***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

Codepage *(USHORT)* — input
Code page:

NULL Use the current-process code page
Other Use the specified code page.

Country *(USHORT)* — input
Country code:

NULL Use the default country code specified in CONFIG.SYS
Other Use the specified country code.

Inchar *(USHORT)* — input
Character to be translated to uppercase.

Outchar *(USHORT)* — return
Translated character:

0 Error occurred
Other The translated character.

Possible returns from WinGetLastError:

PMERR_INVALID_STRING_PARM

Remarks

The case-mapping used is the same as provided by the OS/2 DosCaseMap call.

This call requires the existence of a message queue.

WinValidateRect — Validate Rectangle

This call subtracts a rectangle from the update region of an asynchronous paint window, marking that part of the window as visually valid.

WinValidateRect (hwnd, Rect, IncludeClippedChildren, *Success***)**

Parameters

hwnd *(HWND)* — input
Handle of window whose update region is changed.

If this parameter is HWND_DESKTOP or a desktop-window handle, the function applies to the whole screen (or desktop).

Rect *(RECT)* — input
Rectangle to be subtracted from the window's update region.

Programming Note: The data type *WRECT* may also be used, if supported by the language.

IncludeClippedChildren *(BOOL)* — input
Validation-scope indicator:

TRUE Include descendants of **hwnd** in the valid rectangle
FALSE Include descendants of **hwnd** in the valid rectangle, only if parent is not WS_CLIPCHILDREN.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

The call is not used for CS_SYNCPAINT windows.

This call has no effect on the window if any part of the window has been made invalid since the last call to WinBeginPaint, WinQueryUpdateRect, or WinQueryUpdateRegion.

This call requires the existence of a message queue.

WinValidateRegion — Validate Region

SAA

This call subtracts a region from the update region of an asynchronous paint window, marking that part of the window as visually valid.

WinValidateRegion (hwnd, hrgn, includeClippedChildren, *Success***)**

Parameters

hwnd *(HWND)* — input
Handle of window whose update region is changed.

If this parameter is HWND_DESKTOP or a desktop window handle, the function applies to the whole screen (or desktop).

hrgn *(HRGN)* — input
Handle of subtracted region.

This is the region that is subtracted from the window's update region.

includeClippedChildren *(BOOL)* — input
Validation-scope indicator:

TRUE Include descendants of **hwnd** in the valid region
FALSE Include descendants of **hwnd** in the valid region, only if parent is not WS_CLIPCHILDREN.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND
PMERR_HRGN_BUSY

Remarks

The call is not used for CS_SYNCPAINT windows.

The call has no effect on the window if any part of the window has been made invalid since the last call to WinBeginPaint, WinQueryUpdateRect, or WinQueryUpdateRegion.

This call requires the existence of a message queue.

WinWaitMsg — Wait Message

This call waits for a filtered message.

WinWaitMsg (hab, First, Last, *Success*)

Parameters

hab *(HAB)* — input
Anchor-block handle.

First *(USHORT)* — input
First message identity.

Last *(USHORT)* — input
Last message identity.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Remarks

This call causes the current thread to wait for a message to arrive on the message queue associated with **hab**. This must be the next message since the queue was last inspected by a **Success** return from WinGetMsg or WinPeekMsg. It must also conform to the filtering criteria specified by **First** and **Last**.

For details of the filtering performed by **First** and **Last**, see the WinGetMsg call.

This call requires the existence of a message queue.

WinWindowFromDC — Query Window Handle From Device Context

This call returns the handle of the window corresponding to a particular device context.

```
WinWindowFromDC (hdc, hwnd)
```

Parameters

hdc *(HDC)* — input
Device-context handle.

The device context must first be opened by the WinOpenWindowDC call.

hwnd *(HWND)* — return
Window handle:

NULL Error occurred. For example, the device context has not been opened by the WinOpenWindowDC call.
Other Window handle.

Possible returns from WinGetLastError:

Remarks

This call requires the existence of a message queue.

WinWindowFromID — Query Window Handle From Identifier

This call returns the handle of the child window with the specified identity.

WinWindowFromID (Parent, Identifier, *hwnd*)

Parameters

Parent *(HWND)* — input
Parent-window handle.

Identifier *(USHORT)* — input
Identity of the child window.

hwnd *(HWND)* — return
Window handle:

NULL No child window of the specified identity exists
Other Child-window handle.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

To obtain the window handle for an item within a dialog box, set **Parent** to the dialog-box window's handle, and set **Identifier** to the identity of the item in the dialog template.

WinWindowFromPoint — Window From Point

SAA

This call finds the window below a specified point, that is a descendant of a specified window.

WinWindowFromPoint (Parent, Point, EnumChildren, Lock, *Found***)**

Parameters

Parent *(HWND)* — input
Window handle whose child windows are to be tested:

HWND_DESKTOP	The desktop-window handle, implying that all main windows are tested. In this instance, **Point** must be relative to the bottom left corner of the screen.
Other	Parent-window handle.

Point *(POINT)* — input
The point to be tested.

Specified in window coordinates relative to the window specified by the **Parent** parameter.

EnumChildren *(BOOL)* — input
Test control:

TRUE	Test all the descendant windows, including child windows of child windows
FALSE	Test only the immediate child windows.

Lock *(BOOL)* — input
Lock control:

This parameter is ignored. Window locking is not required in OS/2 Version 1.2 and above.

Found *(HWND)* — return
Window handle beneath **Point**:

NULL	**Point** is outside **Parent**
Parent	**Point** is not inside any of the children of **Parent**
Other	Window handle is beneath **Point**.

Possible returns from WinGetLastError:

PMERR_INVALID_HWND

Remarks

This call only checks the descendants of the specified window.

This call requires the existence of a message queue.

WinWriteProfileData — Write Profile Data

This call writes a string of binary data into the initialization file by putting the data into the area identified by the **AppName / KeyName** parameter pair.

WinWriteProfileData (hab, AppName, KeyName, Value, Size, *Success***)**

Parameters

hab *(HAB)* — input
Anchor-block handle.

AppName *(STRL)* — input
Text string.

AppName is the name of the application for which initialization data is written.

Its length must be less than 1024 bytes including the null termination character. Names starting with the characters "PM_" are reserved for system use. The application name is case-dependent.

KeyName *(STRL)* — input
Text string.

KeyName is the name of the key for which text data is written.

Its length must be less than 1024 bytes including the null termination character. The key name is case-dependent.

This parameter can be NULL in which case **all** the **KeyName / Value** pairs associated with **AppName** are deleted.

Value *(STORAGE)* — input
Value data.

Value is the value of the **KeyName / Value** pair that is written to the file. The string is **not** zero-terminated, and the length of the string is given by **Size**.

If the pointer to **Value** is NULL, the string associated with **KeyName** is deleted.

Size *(USHORT)* — input
Size of value data.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_CAN_NOT_CALL_SPOOLER

WinWriteProfileData – Write Profile Data

Remarks

Because of the binary nature of the data, the input data is not zero-terminated. The length provided is the only way to identify the length of the data.

If the file does not exist, this call creates it, unless this is a delete operation. If the file does exist, but is corrupt, this call returns an error.

If the location of the initialization file means that the file is read-only, no data is written to the file. The user has to copy the file to a read/write area, reissue the call to write data to the file.

This call is similar to PrfWriteProfileData with the **hini** parameter set to HINI_PROFILE. PrfWriteProfileData should be used in preference.

This call requires the existence of a message queue.

WinWriteProfileString — Write Profile String

This call enters a **KeyName / Value** parameter pair into the initialization file, or removes all the **KeyName / Value** pairs for a specified application.

WinWriteProfileString (hab, AppName, KeyName, Value, *Success***)**

Parameters

hab *(HAB)* — input
Anchor block handle.

AppName *(STRL)* — input
Text string.

AppName is the name of the application for which initialization data is written.

Its length must be less than 1024 bytes including the null termination character. Names starting with the characters "PM_" are reserved for system use. The application name is case-dependent.

KeyName *(STRL)* — input
Text string.

KeyName is the name of the key for which text data is written.

Its length must be less than 1024 bytes including the null termination character. The key name is case-dependent.

This parameter can be NULL, in which case **all** the **KeyName / Value** pairs associated with **AppName** are deleted.

Value *(STRL)* — input
Text string.

Value is the value of the **KeyName / Value** pair that is written to the file.

If **Value** is NULL, the string associated with **KeyName** is deleted (that is, the entry is deleted).

If **Value** is not NULL, the string is used as the value of the **KeyName / Value** pair, even if the string has zero length.

Success *(BOOL)* — return
Success indicator:

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError:

PMERR_INVALID_PARM
PMERR_CAN_NOT_CALL_SPOOLER

Remarks

If there is no application field in the file that matches **AppName**, a new application field is created before the **KeyName / Value** entry is made for it.

If the key name does not exist for the application, a new **KeyName / Value** entry is created for that application. If the **KeyName** already exists in the file, the existing value is overwritten.

If the initialization file does not exist, this call creates it. The call can fail if the existing initialization file is corrupt, or it is read-only.

WinWriteProfileString — Write Profile String

For simplicity, any text held in the initialization file should be uppercase (use the WinUpper call). The maximum length of the data string is 64K—1 bytes.

This call is similar to PrfWriteProfileString with the **hini** parameter set to HINI_PROFILE. PrfWriteProfileString should be used in preference.

This call requires the existence of a message queue.

Glossary

This glossary defines terms used in this book. It contains terms specific to the Presentation Manager, but it is not a complete glossary for OS/2 Version 1.2. This glossary includes terms and definitions from the *IBM Dictionary of Computing*, SC20-1699.

A

accelerator. A single key stroke that invokes an application-defined function.

accelerator table. Used to define which key strokes are treated as **accelerators** and the commands they are translated into.

action bar. The area at the top of a window that contains the choices currently available in the application program.

action point. The current position on the screen at which the pointer is pointing. (Contrast with **hot spot** and **input focus.**)

active program. A program currently running on the computer. See also **interactive program, noninteractive program,** and **foreground program.**

active window. The window with which the user is currently interacting.

alphanumeric video output. Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

anchor block. An area of Presentation Manager-internal resources allocated to a process or thread that calls WinInitialize.

anchor point. A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI. American National Standards Institute.

APA. All points addressable.

API. Application programming interface. The formally-defined programming language that is between an IBM application program and the user of the program. See also **GPI.**

area. In computer graphics, a filled shape such as a solid rectangle.

ASCII. American National Standard Code for Information Interchange.

ASCIIZ. A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio. In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous. (1) Without regular time relationship. (2) Unexpected or unpredictable with respect to the execution of a program's instructions.

atom. A constant that represents a string. Once a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an **atom table.** See also **integer atom.**

atom table. Used to relate **atoms** with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also **graphics attributes** and **segment attributes.**

AVIO. Advanced Video Input/Output.

B

background color. The color in which the background of a graphic primitive is drawn.

background mix. An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with **mix.**

Bézier curves. A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier.

bit map. A representation in memory of the data displayed on an APA device, usually the screen.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

bucket. One or more fields in which the result of an operation is kept.

button. A mechanism on a **pointing device,** such as a mouse, used to request or initiate an action. Contrast with **pushbutton** and **radio button.**

C

cached micro presentation space. A presentation space from a Presentation Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

cancel. An action that removes the current window or menu without processing it, and returns the previous window.

cell. See **character cell.**

CGA. Color graphics adapter.

chained list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonym for **linked list**.

character. A letter, digit, or other symbol.

character box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also **character mode**. Contrast with **character cell**.

character cell. The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with **character box**.

character code. The means of addressing a character in a character set, sometimes called **code point**.

character mode. The character mode, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

check box. A control window, shaped like a square button on the screen, that can be in a checked or unchecked state. It is used to select one or more items from a list. Contrast with **radio button**.

check mark. The symbol (√) that is used to indicate a selected item on a pull-down.

child window. A window that is positioned relative to another window (either a main window or another child window). Contrast with **parent window**.

choice. An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol).

class. See **window class**.

class style. The set of properties that apply to every window in a window class.

client area. The area in the center of a window that contains the main information of the window.

clipboard. An area of main storage that can hold data being passed from one Presentation Manager application to another. Various data formats can be stored.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits. The area of the paper that can be reached by a printer or plotter.

clipping path. A clipping boundary in world-coordinate space.

code page. An assignment of graphic characters and control-function meanings to all code points.

code point. Synonym for **character code**.

color dithering. See **dithering**

command line. On a display screen, a display line usually at the bottom of the screen, in which only commands can be entered.

Common Programming Interface. A consistent set of specifications for languages, commands, and calls to enable applications to be developed across all SAA environments. See also **Systems Application Architecture**.

Common User Access. A set of rules that define the way information is presented on the screen, and the techniques for the user to interact with the information.

control. The means by which an operator gives input to an application. A **choice** corresponds to a control.

Control Panel. In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program. The basic function of OS/2 including DOS emulation and the support for keyboard, mouse, and VIO.

control window. A class of window used to handle a specific kind of user interaction. Radio buttons and check boxes are examples.

correlation. The action of determining which element or object within a picture is at a given position on the display. This follows a **pick** operation.

CPI. Common Programming Interface.

current position. The point from which the next primitive will be drawn.

cursor. A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with **pointer** and **input focus**.

D

data structure. (ISO) The syntactic structure of symbolic expressions and their storage-allocation characteristics.

DBCS. See **double-byte character set**.

decipoint. In printing, one tenth of a point. There are 72 points in an inch.

default procedure. Function provided by the Presentation Interface that may be used to process standard messages from dialogs or windows.

default value. A value used when no value is explicitly specified by the user. For example, in the graphics programming interface, the default line-type is 'solid'.

Desktop Manager. In the Presentation Manager, a window which displays a list of groups of programs, each of which can be started or stopped.

desktop window. The window, corresponding to the physical device, against which all other types of windows are established.

device context. A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also **direct device context, information device context, memory device context, metafile device context, queued device context,** and **screen device context.**

device driver. A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space. Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units.

dialog. The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box. A type of window that contains one or more controls for the formatted display and entry of data. Also known as a **pop-up window.** A modal dialog box is used to implement a pop-up window.

Dialog Box Editor. A **WYSIWYG** editor that creates dialog boxes for communicating with the application user.

dialog item. A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog template. The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context. A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also **device context.**

direct manipulation. The action of using the mouse to move objects around the screen. For example, moving files and directories about in the **File Manager.**

directory. A type of file containing the names and controlling information for other files or other directories.

display point. Synonym for **pel.**

dithering. The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work on narrow lines, for example.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. As each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

dragging. In computer graphics, moving an object on the display screen as if it were attached to the pointer.

drawing chain. See **segment chain.**

drop. To fix the position of an object that is being dragged, by releasing the select button of the pointing device.

dynamic segments. Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

E

EBCDIC. Extended binary-coded decimal interchange code.

EGA. Extended graphics adapter.

element. An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

entry field. An area on the screen, usually highlighted in some manner, in which users type information.

entry-field control. The means by which the application receives data entered by the user in an entry field. When it has the input focus, it displays a flashing pointer at the position where the next typed character will go.

exit. The action that terminates the current function and returns the user to a higher level function. Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with **cancel.**

extended-choice selection. A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with **multiple-choice selection.**

F

File Manager. In the Presentation Manager, a program that displays directories and files, and allows various actions on them.

file specification. The full identifier for a file, which includes its file name, extension, path, and drive.

fillet. A curve that is tangential to the end points of two adjoining lines. See also **polyfillet.**

focus. See **input focus.**

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

foreground program. The program with which the user is currently interacting. Also known as **interactive program.**

frame. The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles. Different standard window layouts provided by the Presentation Manager.

full-screen application. An application program that occupies the whole screen.

function key area. The area at the bottom of a window that contains function key assignments such as F1 = Help.

G

Global Descriptor Table (GDT). Defines code and data segments available to all tasks in an application.

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics programming interface. The formally-defined programming language that is between an IBM graphics program and the user of the program. See also **API**.

graphics. A picture defined in terms of graphic primitives and graphics attributes.

graphics attributes. Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also **segment attributes**.

graphics field. The clipping boundary that defines the visible part of the presentation-page contents.

graphics model space. The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as **model space**.

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string. See also **graphics segment**.

graphics segment. A sequence of related graphic primitives and graphics attributes. See also **graphic primitive**.

graying. The indication that a choice on a pull-down is unavailable.

group. A collection of logically-connected controls. For example, the buttons controlling paper size for a printer. See also **program group**.

H

handle. An identifier that represents an object, such as a device or window, to the Presentation Interface.

hard error. An error condition on a network that requires either that the system be reconfigured, or that the source of the error be removed before the system can resume reliable operation.

heap. An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

hit testing. The means of identifying which window is associated with which input device event.

hook. A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

hook chain. A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot. The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with **action point**.

I

icon. A pictorial representation of an item the user can select. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs.

icon area. In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor. The Presentation Manager-provided tool for creating icons.

image font. A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of the symbol. Contrast with **outline font**.

information device context. A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also **device context**.

input focus. The area of the screen that will receive input from an input device (typically the keyboard).

input router. OS/2-internal process that removes messages from the system queue.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

integer atom. A special kind of **atom** that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive program. A program that is running (active) and is ready to receive (or is receiving) input from the user. Compare with **active program** and contrast with **noninteractive program**.

Also known as a **foreground program**.

interchange file. Data that can be sent from one Presentation Interface application to another.

J

journal. A special-purpose file that is used to record changes made in the system.

K

Kanji. A graphic character set used in Japanese ideographic alphabets.

kerning. The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

L

label. In a graphics segment, an identifier of one or more elements that is used when editing the segment.

language support procedure. Function provided by the Presentation Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

LIFO stack. A data stack from which data is retrieved in last-in, first-out order.

linked list. Synonym for **chained list.**

list box. A control window containing a vertical list of selectable descriptions.

Local Descriptor Table (LDT). Defines code and data segments specific to a single task.

M

main window. The window that is positioned relative to the **desktop window.**

marker box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol. A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

maximize. A window-sizing action that makes the window the largest size possible.

media window. The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory device context. A logical description of a data destination that is a memory bit map. See also **device context.**

menu. A type of panel that consists of one or more selection fields. Also called a **menu panel.**

message. A packet of data used for communication between the Presentation Interface and windowed applications.

message filter. The means of selecting which messages from a specific window will be handled by the application.

message queue. A sequenced collection of messages to be read by the application.

metafile. The generic name for the definition of the contents of a picture. Metafiles are used to allow pictures to be used by other applications.

metafile device context. A logical description of a data destination that is a metafile, which is used for graphics interchange. See also **device context.**

metalanguage. A language used to specify another language. In this publication, the data types are described using a metalanguage so as to make the descriptions independent of any one computer language.

micro presentation space. A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize. A window-sizing action that makes the window the smallest size possible. In the Presentation Manager, minimized windows are represented by icons.

mix. An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as **foreground mix.** Contrast with **background mix.**

mixed character string. A string containing a mixture of one-byte and **Kanji** or Hangeul (two-byte) characters.

mnemonic. A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item.

modal dialog box. The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also known as a **serial dialog box.** Contrast with **parallel dialog box.**

modeless dialog box. The type of control that allows the operator to perform input operations on any of the application's windows. Also known as a **parallel dialog box.** Contrast with **modal dialog box.**

model space. See **graphics model space.**

mouse. A hand-held device that is moved around to position the pointer on the screen.

multiple-choice selection. A mode that allows users to select any number of choices, including none at all. See also **check box.** Contrast with **extended-choice selection.**

multitasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

N

named pipe. A named object that provides client-to-server, server-to-client, or full duplex communication between unrelated processes. Contrast with **unnamed pipe.**

noninteractive program. A program that is running (active) but is not ready to receive input from the user. Compare with **active program,** and contrast with **interactive program.**

nonretained graphics. Graphic primitives that are not remembered by the Presentation Interface once they have been drawn. Contrast with **retained graphics**.

null-terminated string. A string of (n + 1) characters where the (n + 1)th character is the 'null' character (X'00'), and is used to represent an n-character string with implicit length. Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object window. A window that does not have a parent, but which may have child windows. An object window cannot be presented on a device.

open. To start working with a file, directory, or other object.

outline font. A set of symbols, each of which is created as a series of lines and curves. Contrast with **image font**.

output area. The area of the output device within which the picture is to be displayed, printed, or plotted.

owner window. A window into which specific events that occur in another (owned) window are reported.

P

page viewport. A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint. The action of drawing or redrawing the contents of a window.

panel. A particular arrangement of information grouped together for presentation to the user in a window.

panel body area. The part of a window not occupied by the action bar or function key area. The panel body area may contain information, selection fields, and entry fields. Also known as **client area**.

panel body area separator. A line or color boundary that provides users with a visual distinction between two adjacent areas of a panel.

panel title. A panel element that identifies the information in the panel.

paper size. The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box. See **modeless dialog box**.

parent window. The window relative to which one or more child windows are positioned. Contrast with **child window**.

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for **display point**, **pixel**, and **picture element**.

pick. To select part of a displayed object using the pointer.

picture chain. See **segment chain**.

picture element. Synonym for **pel**.

pipe. See **named pipe**, **unnamed pipe**.

pixel. Synonym for **pel**.

plotter. An output device that uses pens to draw its output on paper or transparency foils.

pointer. The symbol displayed on the screen that is moved by a pointing device, such as a **mouse**. The pointer is used to point at items that users can select. Contrast with **cursor**.

pointing device. A device (such as a mouse) used to move a pointer on the screen.

pointings. Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a **mouse**.

polyfillet. A curve based on a sequence of lines. It is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also **fillet**.

polyline. A sequence of adjoining lines.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with **push**.

pop-up window. A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window.

Presentation Manager. The visual component of OS/2 that presents, in windows, a graphics-based interface to applications and files installed and running in OS/2.

presentation page. The coordinate space in which a picture is assembled for display.

presentation space (PS). Contains the device-independent definition of a picture.

primary window. The window in which the main dialog between the user and the program takes place. See also **secondary window**.

primitive. See **graphic primitive**.

primitive attribute. A specifiable characteristic of a graphic primitive. See **graphics attributes**.

print job. The result of sending a document or picture to be printed.

Print Manager. In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

process. An instance of an executing application and the resources it is using.

program details. Information about a program that is specified in the **Program Manager** window and is used when the program is started.

program group. In the Presentation Manager, several programs that can be acted upon as a single entity.

program name. The full file specification of a program. Contrast with **program title.**

program title. The name of a program as it is listed in the **Program Manager** window. Contrast with **program name.**

pull-down. An **action bar** extension that displays a list of choices available for a selected action bar choice.

push. To add an item to a last-in-first-out stack of items. Contrast with **pop.**

pushbutton. A control window, shaped like a round-cornered rectangle and containing text, that invokes an immediate action, such as 'enter' or 'cancel'.

Q

queue. A list of print jobs waiting to be printed.

queued device context. A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also **device context.**

R

radio button. A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from list. Contrast with **check box.**

realize. To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

refresh. To update a window, with changed information, to its current status.

region. A clipping boundary in device space.

resource. The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

restore. To return a window to its original size or position following a sizing or moving action.

retained graphics. Graphic primitives that are remembered by the Presentation Interface after they have been drawn. Contrast with **nonretained graphics.**

reverse video. A form of alphanumeric highlighting for a character, field, or cursor, in which its color is exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB. Red-green-blue. For example "RGB display."

roman. Relating to a type style, with upright characters.

S

SAA. Systems Application Architecture.

screen. The physical surface of a work station or terminal upon which information is presented to users.

screen device context. A logical description of a data destination that is a particular window on the screen. See also **device context.**

scroll bar. A control window, horizontally or vertically aligned, that allows the user to scroll additional data into an associated panel area.

scrollable entry field. An entry field larger than the visible field.

scrollable selection field. A selection field that contains more choices than are visible.

scrolling. Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window. A type of window associated with the primary window in a dialog. A secondary window begins a secondary and parallel dialog that runs at the same time as the primary dialog.

segment. See **graphics segment.**

segment attributes. Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain. All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for **picture chain.**

segment priority. The order in which segments are drawn.

segment store. An area in a normal graphics presentation space where retained graphics segments are stored.

select. To mark or choose an item. Note that **select** means to mark or type in a choice on the screen; **enter** means to send all selected choices to the computer for processing.

select button. The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor. A type of cursor used to indicate the choice or entry field users want to interact with. It is represented by highlighting the item it is currently positioned on.

selection field. A field containing a list of choices from which the user can select one or more.

semaphore. An object used by multi-threaded applications for signaling purposes and for controlling access to serially reusable resources.

separator. See **panel body area separator.**

serial dialog box. See **modal dialog box.**

serially reusable resource (SRR). A logical resource or object that can be accessed by only one task at a time.

session. A routing mechanism for user interaction via the console.

shadow box. The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shear. The tilt of graphics text when each character leans to the left or right while retaining a horizontal baseline.

Shutdown. The procedure required before the computer is switched off to ensure that data is not lost.

sibling windows. Child windows that have the same parent window.

slider box. An area on the scroll bar that indicates the size and position of the visible information in a panel area in relation to the information available. Also known as **thumb mark.**

spline. A sequence of one or more Bézier curves.

spooler. A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete, and the required device is available. The spooler prevents output from different sources being intermixed.

standard window. A collection of windows that form a panel.

static control. The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

style. See **window style.**

suballocation. The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

switch. An action that moves the input focus from one area to another. This can be within the same window or from one window to another.

switch list. See **Task List**

symbolic identifier. A text string that equates to an integer value in an include file, that is used to identify a programming object.

System Menu. In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system queue. This is the master queue for all pointer device or keyboard events.

Systems Application Architecture. A formal set of rules that enables applications to be run without modification in different computer environments.

T

tag. One or more characters attached to a set of data that defines the formatting or other characteristics of the set, including its definition.

Task list. In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

template. An ASCII-text definition of an action bar and pull-down menu, held in a resource file, or as a data structure in program memory.

text. Characters or symbols.

text cursor. A symbol displayed in an entry field that indicates where typed input will appear.

text window. Also known as the VIO window.

thread. A unit of execution within a process.

thumb mark. The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a **slider box.**

tilde. A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

title bar. The area at the top of a window that contains the window title. The title bar is highlighted when that window has the input focus. Contrast with **panel title.**

transform. (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a **transformation.**

Tree. In the Presentation Manager, the window in the **File Manager** that shows the organization of drives and directories.

U

unnamed pipe. A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with **named pipe.**

update region. A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window, that are visually invalid or incorrect, and therefore in need of repainting.

User Shell. A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

V

VGA. Video graphics array.

viewing pipeline. The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window. Clipping boundary that defines the visible part of model space.

VIO. Video Input/Output.

virtual memory (VM). Addressable space that is apparent to the user as the processor storage space, but not having a fixed physical location.

visible region. A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

W

wild-card character. The global file-name characters ? or *.

window. A rectangular area of the screen with visible boundaries in which information is displayed. A window may be smaller than or equal to the screen. Windows may overlap on the screen and give the appearance of one window being on top of another.

window class. The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates. The means by which a window position or size is defined; measured in device units, or **pels**.

window procedure. Code that is activated in response to a message.

window rectangle. The means by which the size and position of a window is described in relation to the desktop window.

window style. The set of properties that influence how events related to a particular window will be processed.

workstation. A display screen together with attachments such as a keyboard, a local copy device, or a tablet.

world coordinates. Application-convenient coordinates used for drawing graphics.

world-coordinate space. Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG. What You See Is What You Get. A capability that enables text to be displayed on a screen in the same way it will be formatted on a printer.

Z

z-order. The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming. In graphics applications, the process of increasing or decreasing the size of picture.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

IBM United Kingdom
International Products Limited
PO Box 41, North Harbour
Portsmouth, PO6 3AU
England

Printed in Denmark by Rosendahl - Esbjerg